500 Tips to Use DirectX Better!

DirectX

Visual C++ 2002/2003/2005対応

▶逆引き大全◀
500の極意

鎌田茂雄
著

秀和システム

■注意

(1)　本書は著者が独自に調査した結果を出版したものです。
(2)　本書は内容について万全を期して制作いたしましたが、万一、ご不審な点や誤り、記載漏れなどお気付きの点がありましたら、出版元まで書面にてご連絡ください。
(3)　本書の内容に関して運用した結果の影響については、上記　項にかかわらず、責任を負いかねます。あらかじめご了承ください。
(4)　本書の一部または全部を著作権法の定める範囲を超え、出版元から文書による承諾を得ずに無断で複写、複製、転載することを禁じます。
(5)　本書に記載されているホームページのアドレスなどは、予告なく変更されることがあります。
(6)　商標
本書に記載されている会社名、商品名などは一般に各社の商標または登録商標です。なお本書では™、©、®を割愛しています。

はじめに

最初に秀和システムさんから本書の話があった時、筆者は安易に考えてしまいました。項目数は多いものの“リファレンス”であるので、さほど困難ではないだろうと。

ところが、すぐにその認識が間違っていると気付くことになります。一般的な逆引き本のように“ソフトウェアの操作”に関する逆引きを主にイメージしてしまっていたのです。しかし、DirectXに限らず“API”の使用方法は無数の用途があり、操作と結果が1体1に対応しているソフトウェアとは異なり、「このボタンを押せば、こうなります」というようなわけにはいかないのです。APIは逆引きは馴染みません。それに気付いたときには既にプロジェクトが始動していたので「これはかなり厄介だぞ」と思ったわけです。他の仕事を一時休止してまで本書の作業に取り組み、最善を尽くしました。それでも出版までにほとんど1年かかるほど本書のボリュームは膨大です。

例えば、それぞれの項目を単に“ヘルプの逆”にしたような本は誰も要らないのです。そのような安直なコンセプトだとすると、本書は全くもって価値のないただの紙になってしまいます。しかも800ページにも届くような無駄な紙の塊です。なぜなら、そんなことはヘルプの単語検索機能で検索すればいいだけの話で、それこそまさに最高の逆引きでしょう。

もちろん本書はそのような設計思想ではありません。ヘルプではカバーできない領域を解説し、本書固有の意義を見出しています。

本書は目的主導で構成されています。例えば、ビジュアル関係では「メッシュをレンダリングするには？」とか「セルフシャドウを付けるには？」、サウンド関係では「ストリーミング再生をしたい」、「スピーカーから出る音を録音するのは？」など、さらにDirectSetupを取り上げた書籍は、少なくとも筆者の記憶では本書が初めてだと思います。およそ開発者が望むであろうあらゆる要求に対してコードを書いたつもりです。

このようにある程度大きな目的からAPIを探そうとすれば、ヘルプは答えを返してくれません。実際本書の項目は、ヘルプを見ただけでは作成できないものが多くあります。場合によっては、ヘルプにはまったく載っていない技術も解説しています。その結果、“逆引き本”とは思えないほどページ数を割いている項目もあります。

なお、開発及び本書サンプルのビルドにはDirectX9.0C SDKが必要ですが、SDKは読者の側で用意してください。インターネット環境があれば無料で手に入ります。インターネット環境に無い場合は、マイクロソフトからCD-ROMを購入することもできます。価格は千数百円程度だったと思いますが、何れにしてもマイクロソフトに問い合わせて下さい。本書はそこまでサポートするものではありません。そもそも本書を購入する方は既にSDKをインストールしている方がほとんどだと思うので、その辺りのことは本書の範囲外とさせていただきます。したがって、コンパイラーやSDKの“設定方法”や“設定ミス”に関る質問には一切お答えしません。

本書のサンプルは、DirectXの4つのコンポーネントについて合計279種類作成しまし

た。サンプルはVisual C++で作成したので、Visual C++のプロジェクトファイル及びソリューションファイルと共にプロジェクトフォルダ単位で収録していますが、ソースコードさえあれば他のコンパイラーでも使用できることは言うまでもありません。プロジェクトを作成する僅かな手間がかかるだけで、ソースコードさえあればいいわけです。

収録しているVisual C++用のプロジェクト及びソリューションファイルは次の3つのバージョンです。

VC7.0（.NET2002）
VC7.1（.NET2003）
VC8.0（.NET2005）

もちろん、プロジェクトを開くときは読者のVCバージョンに合ったソリューションファイルあるいはプロジェクトファイルをクリックしてください。

1つのサンプルに3つのバージョン用のソリューションファイルとプロジェクトファイルを詰め込んでいるので、プロジェクトフォルダ内には最低でも6つのファイルのファイルが存在することになり、一見するとファイル数が多く感じることがあるかもしれませんが、本書のサンプルの多くはソースファイルが一つだけの非常にコンパクトなものです。

本書の目標は「ボロボロになるまで」活用されることです。ページがほころびるほど何回も開かれる、「開発者にとって必携の書」になることを期待しています。

本書の読み方

ここでは、本書で使用している表記、アイコン等について説明しています。

極意(Tips)を一言で説明

プログラミングの考え方や手順、使用するメソッドなど一言で説明しています。

極意(Tips)の番号

目次で見つけた「極意(Tips)」をすぐに見つけることができます。

プログラミング上の要望・質問

「～したい」「～するには」といった要望や質問を示しています。自分のやりたいことを探してください。

極意(Tips)のレベル

レベルには「初級」「中級」「上級」の3レベルがあります。テクニックの難易度の目安にしてください。

サンプルのプロジェクト名

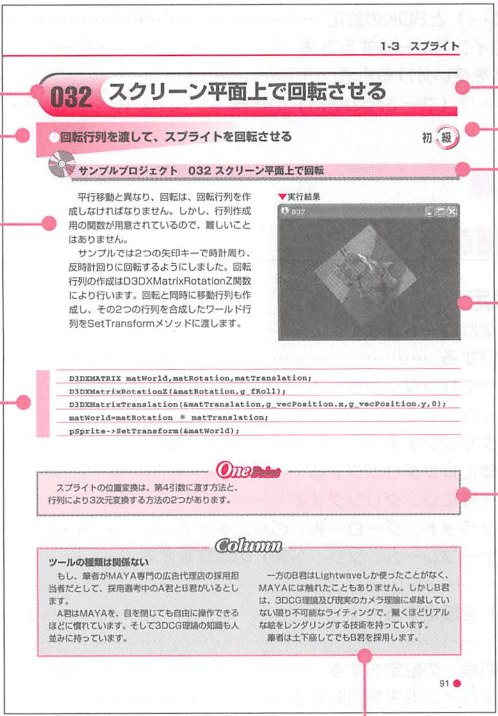
サンプルファイルを収録したフォルダの名称です。詳しくはCD-ROMをご覧ください。

画面

実際のプログラムの参考になるように、サンプル実行後の画面などを示しています。

OnePoint

この極意(Tips)の補足説明を示しています。



1-3 スプライト

032 スクリーン平面上で回転させる

回転行列を渡して、スプライトを回転させる

初級

サンプルプロジェクト 032 スクリーン平面上で回転

平行移動と異なり、回転は、回転行列を作成しなければなりません。しかし、行列作成用の関数が用意されているので、難しいことはありません。

サンプルでは2つの矢印キーで時計周り、反時計回りに回転するようにしました。回転行列の作成はD3DXMatrixRotationZ関数により行います。回転と同時に移動行列も作成し、その2つの行列を合成したワールド行列をSetTransformメソッドに渡します。

▼実行結果

```
D3DXMATRIX matWorld,matRotation,matTranslation;
D3DXMatrixRotationZ(&matRotation,g_fRoll);
D3DXMatrixTranslation(&matTranslation,g_vecPosition.x,g_vecPosition.y,0);
matWorld=matRotation * matTranslation;
pSprite->SetTransform(&matWorld);
```

One Point

スプライトの位置変換は、第4引数に渡す方法と、行列により3次元変換する方法の2つがあります。

Column

ツールの種類は関係ない

もし、筆者がMAYA専門の広告代理店の採用担当者だとして、採用選考中のA君とB君がいるとします。

A君はMAYAを、目を閉じても自由に操作できるほどに慣れています。そして3DCG理論の知識も人並みに持っています。

一方のB君はLightwaveしか使ったことがなく、MAYAには触れたこともありません。しかしB君は、3DCG理論及び現実のカメラ理論に卓越していない限り不可能なライティングで、驚くほどリアルな絵をレンダリングする技術を持っています。

筆者は土下座してでもB君を採用します。

91

リスト

サンプルのコードなどを示しています。

COLUMN

DirectXで知っておきたい知識などを簡潔にまとめてあります。

極意(Tips)の詳細

この極意(Tips)を詳しく説明しています。手順は、ステップを追って実行できるようになっています。

目次 -CONTENTS-

はじめに ……3
本書の読み方 ……5

序章　準備

001　DirectXによるゲーム開発について ……22
002　必要なものについて ……24
003　コンパイラー（Visual C++）とSDKの設定 ……26
004　古いバージョンのSDKをインストールする方法 ……30
005　異なるバージョンのSDKを使い分ける方法 ……35
006　本書におけるフレームワーク（コード全の構造）の特徴 ……37

第1部　Direct3Dの極意

第1章　レンダリングの極意

1-1　レンダリング準備と設定

007　レンダリングに最低限必要なコードを書く ……42
008　ウィンドウにレンダリングする ……45
009　フルスクリーンウィンドウにレンダリングする ……46
010　ダイアログ全体にレンダリングする ……48
011　ダイアログの一部にレンダリングする ……51
012　モードレスダイアログ全体にレンダリングする ……53
013　モードレスダイアログの一部にレンダリングする ……56
014　シェーディングモード（フラット、グーロー等）の設定をする ……59
015　塗りつぶしモード（ワイヤーフレーム、ソリッド等）の設定をする ……60
016　カリングの設定をする ……61
017　アンビエント光（環境光）を設定する ……63
018　いろいろなライトを作る ……64
019　アルファブレンド（透過処理）の設定をする ……69
020　フルスクリーンにおけるデバイス消失を復旧する ……70

1-2　レンダリングの基本的な流れ

021　画面をクリアする ……74
022　シーンを開始・終了する ……74
023　ワールド変換をする ……75
024　ビュー変換をする ……79
025　プロジェクション変換をする ……81
026　画面のフリップとは（画面更新をモニターに反映させる） ……83

1-3　スプライト

027　スプライトとは ……84
028　スプライトにする画像の作成について ……85

<ID3DXSpriteによるスプライト>

029　スプライト（及びテクスチャ）オブジェクトを作成する ……86
030　スクリーン平面上にレンダリングする ……87
031　スクリーン平面上で平行移動する ……89
032　スクリーン平面上で回転させる ……91
033　スクリーン平面上で拡大縮小する ……92
034　スクリーン平面上で半透明にする ……93
035　スクリーン平面上で“ふち”を背景色に馴染ませる ……94
036　スクリーン平面上で意図した重ね順で重ねる ……96
037　ワールド空間内にレンダリングする ……99
038　ワールド空間内で平行移動する ……100
039　ワールド空間内で回転させる ……102
040　ワールド空間内で拡大縮小する ……103
041　ワールド空間内で半透明にする ……104
042　ワールド空間内で“ふち”を背景色に馴染ませる ……106
043　ワールド空間内で意図した重ね順で重ねる ……107
044　メッシュと一緒にレンダリングする ……108
045　フルスクリーンに復帰した時に再初期化をする ……110

<板ポリゴンで作成するスプライト（テクスチャードクアッド）>

046　テクスチャードクアッドを作成する ……112
047　テクスチャードクアッドをレンダリングする ……113
048　テクスチャードクアッドを平行移動する ……115
049　テクスチャードクアッドを回転させる ……116
050　テクスチャードクアッドを拡大縮小する ……117
051　テクスチャードクアッドを半透明にする ……118
052　テクスチャードクアッドの“ふち”を背景色に馴染ませる ……119
053　テクスチャードクアッドを意図した重ね順で重ねる ……120
054　メッシュと一緒にレンダリングする ……122
055　フルスクリーンに復帰時に再初期化をする ……124

1-4　3Dジオメトリ

<3Dプリミティブ>

056　プリミティブとは ……126
057　ポイント(点)を作成する ……126
058　ポイント（点）のレンダリング ……128
059　ライン（線）を作成する ……130
060　ライン（線）のレンダリング ……131
061　ポリゴン（面）を作成する ……132
062　ポリゴン（面）のレンダリング ……134
063　プリミティブのマテリアル（色、光沢等）を設定する ……136
064　プリミティブを半透明でレンダリングする ……140
065　テクスチャを貼ったプリミティブをレンダリングする ……141

066 プリミティブを移動する ……142
067 プリミティブを回転させる ……144
068 プリミティブを拡大縮小する ……145
069 1つのプリミティブを複数レンダリングする ……146
070 複数のプリミティブをレンダリングする ……147
071 フルスクリーンに復帰した時に再初期化をする ……149
072 プリミティブをビルボーディングする ……151

<メッシュ>

073 メッシュとは ……153
074 メッシュの作成について ……154
075 スフィア（球）を作成する ……155
076 スフィア（球）のレンダリング ……157
077 ボックス（立方体）を作成する ……159
078 ボックス（立方体）のレンダリング ……159
079 シリンダー（円柱）を作成する ……161
080 シリンダー（円柱）のレンダリング ……163
081 トーラス（ドーナツ形）を作成する ……165
082 トーラス（ドーナツ形）のレンダリング ……166
083 ティーポットを作成する ……168
084 ティーポットのレンダリング ……169
085 文字ジオメトリを作成する ……171
086 文字ジオメトリのレンダリング ……172
087 メッシュをXファイルから読み込む ……174
088 メッシュをレンダリングする ……176
089 メッシュを半透明でレンダリングする ……177
090 テクスチャなしメッシュにテクスチャ座標を追加する ……179
091 メッシュを移動する ……181
092 メッシュを回転させる ……182
093 メッシュを拡大縮小する ……184
094 メッシュをビルボーディングする ……185
095 1つのメッシュを複数レンダリングする ……186
096 複数のメッシュをレンダリングする ……187
097 フルスクリーンに復帰した時に再初期化をする ……188
098 プログレッシブメッシュをレンダリングする ……191
099 プログレッシブメッシュを作成する ……196

<アニメーションメッシュ>

100 アニメーションメッシュとは ……200
101 アニメーションメッシュの作成について ……201
102 アニメーションメッシュをXファイルから読み込む ……203
103 フレーム階層を展開・走査する ……204
104 階層展開（アロケート・ヒエラルキー）クラスのメソッドを実装する ……206
105 アニメーションメッシュをレンダリングする ……211
106 アニメーションスピードを調整する ……213
107 時間指定によりモーションを切り替える ……214
108 トラック切り替えによりモーションを切り替える ……216

<スキンメッシュ>

109　スキンメッシュとは ……222
110　スキンメッシュの作成について ……222
111　スキンメッシュをXファイルから読み込む ……224
112　スキンメッシュをレンダリングする ……226
113　スキンアニメーションスピードを調整する ……229
114　時間指定によりモーションを切り替える ……230
115　トラック切り替えによりモーションを切り替える ……232

<パーティクル>

116　ポイントスプライトとは ……234
117　ポイントスプライトをレンダリングする（カメラ空間）……235
118　ポイントスプライトレンダリングする（スクリーン空間）……236
119　ポイントスプライトのサイズを変更する（カメラ空間）……238
120　ポイントスプライトのサイズを変更する（スクリーン空間）……240
121　ポイントスプライトをパーティクルらしくレンダリングする ……242

1-5　ジオメトリの質感作成・調整

<シェーディングモデル>

122　メッシュの頂点のみレンダリングする ……245
123　メッシュをワイヤーフレームでレンダリングする ……246
124　フラットシェーディングする ……247
125　グーローシェーディングする ……248
126　フォンシェーディングする ……249
127　トゥーン（セル画風）シェーディングする ……250

<陰影と光沢>

128　固定機能シェーダーで陰影を付ける ……258
129　プログラマブルシェーダーで陰影を付ける ……259
130　固定機能シェーダーで光沢を付ける ……262
131　プログラマブルシェーダーで光沢を付ける ……264
132　アイススティックの表面のような霜（しも）を付ける ……266

<マテリアル（色、光沢等）>

133　マテリアルとは ……270
134　マテリアルの作成と設定 ……270
135　プラスチックのような質感にする ……275
136　ゴムのような質感にする ……276
137　金属のような質感にする ……277
138　半透明にする ……279

<テクスチャ>

139　テクスチャとは ……280
140　テクスチャの作成と初期化 ……280
141　テクスチャのフォーマットとメモリサイズを理解する ……282
142　ミップマップとメモリサイズを理解する ……284
143　ミップマップの深さを変更する ……285
144　アルファ付きテクスチャで透過部分を作る ……285
145　マルチパスとは ……287

146　マルチテクスチャとは ……………………………………………………………288
147　テクスチャステージとは …………………………………………………………288
148　ブレンドできるテクスチャの数を調べる ………………………………………289
149　2の累乗サイズしかサポートしていないか調べる ………………………………290
150　正方形しかサポートしていないか調べる ………………………………………291
151　プリミティブに2枚のテクスチャをブレンドして貼る …………………………292
152　プリミティブに8枚のテクスチャをブレンドして貼る …………………………293
153　メッシュに2枚のテクスチャをブレンドして貼る ………………………………296
154　メッシュに8枚のテクスチャをブレンドして貼る ………………………………297
155　マルチパスでテクスチャをブレンドする ………………………………………300
156　テクスチャのないメッシュにテクスチャを貼る …………………………………302
157　テクスチャ座標を自動生成する …………………………………………………303
158　テクスチャ自体を動かす …………………………………………………………305
159　スプライトのテクスチャを反転させる ……………………………………………308
160　プログラマブルシェーダーでテクスチャを貼る …………………………………310
161　マルチテクスチャでメッシュを「燃やす」…………………………………………312
162　プログラマブルシェーダーにおけるテクスチャの合成 …………………………314
163　プログラマブルシェーダーで「燃やす」……………………………………………316
164　プログラマブルシェーダーで「焦がす」……………………………………………318

<凹凸感>

165　バンプマッピング …………………………………………………………………320
166　プログラマブルシェーダーで海水面を作る ………………………………………323

<周囲（環境）の映りこみ>

167　スフィアマップで擬似的に映りこませる …………………………………………325
168　スフィアテクスチャを作成する ……………………………………………………327
169　キューブマップでリアルに映りこませる …………………………………………328
170　キューブテクスチャ用の画像を作成する …………………………………………332
171　キューブテクスチャを作成・初期化する …………………………………………335

<プログラマブルシェーダー共通基本処理>

172　最小機能で動くシェーダーを書く …………………………………………………336
173　ディフューズ色くらいは反映したい ………………………………………………339
174　陰影くらいは付けたい ……………………………………………………………340
175　シェーダーにデータを渡す …………………………………………………………342
176　シェーダーからデータを取得する …………………………………………………343
177　テクスチャとサンプリングを設定する ……………………………………………344

<アルファブレンディング>

178　アルファブレンディングする ………………………………………………………346
179　カメラ位置に依存しないアルファブレンディングを実現する ……………………347
180　マテリアルのアルファ成分を反映させる …………………………………………349
181　アルファ付きテクスチャの特定領域だけを透過させる …………………………350
182　透過している裏ポリゴンを自然にする（カリングトリック）……………………351

1-6　文字のレンダリング

183　情報確認用の文字列を描画する ……………………………………………………353
184　フォント名の一覧ってないの？ ……………………………………………………355

1-7　シーンの設定

＜カメラワーク＞
185　カメラとは……356
186　カメラを移動してシーンを移動させる……357
187　カメラを回転してシーンを回転させる……358
188　特定の物体を注視させる……360
189　フォロートラックを行う（注視＋追従移動）……361
190　回り込みトラックを行う（注視＋移動＋回転）……362
191　クレーンを行う（注視＋移動）……363
192　ズームイン・アウトする……365
193　ドリーイン・アウトを行う（注視＋移動＋ズーム）……366
194　カメラを画面に対して絶対的に回転させる……368
195　カメラを画面に対して絶対的に回転させる②（クオータニオン版）……370
196　カメラに対し一定なジオメトリを作る（ビルボーディング）……373
197　3人称視点（TPV）……375
198　1人称視点（FPV)⇔3人称視点（TPV）相互切り替え……376
199　ビュー変換行列を自分で作る……377
＜マルチビュー＞
200　1つのビューを分割した領域にレンダリングする……378
201　異なるビューを分割した領域にレンダリングする……380
202　バックミラーを表現する……382
203　1つのビューを別のウィンドウにレンダリングする……384
204　異なるビューを別のウィンドウにレンダリングする……388

1-8　シーンの質感作成

＜フォグ＞
205　フォグとは？……390
206　頂点フォグの問題点……392
207　開始点と終了点によりフォグをかける……393
208　可視領域内における密度によりフォグをかける……394
209　頂点フォグ……395
210　ピクセルフォグ（指数的）……396
＜ぼかし＞
211　フルシーンアンチエイリアシング（FSAA）のサポートの有無を調べる……397
212　フルシーンアンチエイリアシング（FSAA）をかける……398
213　実用的モーションブラー……399
214　シーン全体をぼかす……402
215　距離によりぼかし具合を変化させる……405
216　被写界深度（DOF）効果を出す……408
＜影＞
217　平面投影法で影を作る……410
218　シャドウボリューム法で影を付ける……412
219　ポイントライト時のシャドーボリューム……417
220　シャドウボリューム自体をレンダリングする……418
221　ダブルステンシルでシャドウボリュームを高速化する……419

222　シャドウマップ法で影を付ける ……420
223　シャドウマップ法で影を付ける（ライティングを加味）……425
224　シャドウマップ法で影を付ける（ライティング、テクスチャを加味）……428
225　深度テクスチャを表示してみる ……431

第2章　ダイナミクスの極意

2-1　レイによる衝突判定

<直線と面の交差検出>

226　直線と無限遠平面の交差を検出する ……434
227　平面方程式を求める ……437
228　交点が平面の内か外かを検出する ……439
229　直線と有限遠平面の交差を検出する ……440
230　直線と平面の交点座標を算出する ……441

<直線とメッシュの交差検出>

231　直線（レイ）とメッシュの交差を検出する ……442
232　“ワールド系での”直線（レイ）とメッシュの交差を検出する……443
233　交差しているメッシュ内のポリゴンを得る ……445
234　交差しているメッシュ内ポリゴンの平面方程式を計算する ……446
235　線（レイ）とメッシュ（内ポリゴン）の交点座標を求める ……448
236　直線（レイ）とアニメーションメッシュの交差を検出する ……449
237　交差しているアニメーションメッシュ内のパーツメッシュを得る ……451
238　交差しているアニメーションメッシュ内のポリゴンを得る ……452
239　交差しているアニメーションメッシュ内ポリゴンの平面方程式を計算する ……453
240　線（レイ）とアニメーションメッシュ（内ポリゴン）の交点座標を求める ……454

2-2　バウンディングスフィアによる衝突判定

241　バウンディングスフィアとは？ ……455
232　バンディングスフィアにより衝突判定する ……456
243　バウンディングスフィアにより衝突判定する（アニメーションメッシュ）……459

2-3　バウンディングボックスよる衝突判定

<基底軸平行バウンディングボックスAABB>

244　AABBとは？……461
245　メッシュのAABBを作成する……462
246　AABBにより衝突判定する ……464

<有向バウンディングボックスOBB>

247　OBBとは？……465
248　OBBにより衝突判定する……466
249　OBBツリーにより衝突判定する（アニメーションメッシュ） ……470

2-4　落下と跳ね返り

250　自由落下 ……471
251　跳ね返らせる ……473

第3章　バッファ、サーフェイス、Xファイルの極意

3-1　メッシュ関連のバッファ

<頂点バッファ>

252　メッシュ内の頂点バッファにアクセスする……476
253　メッシュ内の頂点数を取得する……477
254　メッシュ内のポリゴン数を取得する……477
255　頂点バッファから頂点情報を取得する……478
256　メッシュの頂点に接線情報を加える……479

<インデックスバッファ>

257　インデックスバッファとは？……480
258　インデックスバッファにアクセスする……481
259　インデックスによりポリゴンの3頂点を取得する……481
260　ポリゴンの頂点法線を取得する……483
261　ポリゴンの頂点法線から面法線を算出する……485

<属性バッファと属性テーブル>

262　属性バッファとは？……486
263　属性テーブルとは？……487
264　属性バッファにアクセスする……488
265　属性テーブルを取得する……489
266　属性バッファを表示する……490
267　属性テーブルを表示する……491
268　属性の活用例（DrawSubsetに依らないサブセットごとのレンダリング）……492

<隣接性バッファ>

269　隣接性バッファとは？……494
270　隣接性バッファの作成……494
271　隣接性バッファの表示……495
272　隣接性バッファの活用例……496

3-2　Xファイル

273　Xファイルとは？……498
274　Xファイルの構造を理解する……498
275　Xファイル自体はどうやって作るの？……504
276　Xファイルのサイズを小さくする……505
277　メッシュの頂点数を削減してXファイルを小さくする……506
278　テクスチャが貼られない場合の原因は？……507
279　同じメッシュでもポリゴン数が違うのはなぜ？……508
280　Xファイルにするとポリゴンが欠けたりするのはなぜ？……508
281　Xファイルに複数のアニメーション情報を収録する……509

3-3　リソース管理、各種サーフェイス

<サーフェイス>

282　プールとは？……509
283　プールの種類とそれぞれの意味は？……510
284　ロックとは？……511

285 ロック時のフラグの意味は？ ……513

＜テクスチャ（サーフェイスのコンテナ）＞

286 ダーティ領域とは？ ……514
287 ダーティ領域を設定する ……516
288 テクスチャをロックする ……518
289 ビデオメモリにあるテクスチャをロックする ……519
290 テクスチャ内のピクセルを変更する ……519
291 テクスチャ内のピクセルを参照する ……521
292 テクスチャからテクスチャへピクセルをコピーする ……521
293 テクスチャを単色で塗りつぶす ……522
294 テクスチャ内のイメージを動かす ……523

＜画面バッファ関連（サーフェイス）＞

295 フロントバッファ、バックバッファとは？ ……524
296 深度バッファ（Zバッファ）とは？ ……525
297 ステンシルバッファとは？ ……526

＜レンダリングターゲット＞

298 レンダリングターゲットを変更する ……526
299 テクスチャ上にレンダリングする ……527
300 空のオフスクリーンサーフェイスを作成する ……530
301 フルスクリーン時に表示されているピクセルデータを取得する ……531
302 デスクトップに表示されているピクセルデータを取得する ……533
303 ウィンドウクライアント画面のピクセルデータを取得する ……534

第4章　一歩進んだDirect3Dの極意

4-1　一歩進んだDirect3Dの極意

304 ビデオカード及びドライバ関係の能力を取得する ……536
305 自分の環境でのシェーダーの情報を取得する ……541
306 デスクトップのサイズ、フォーマットを取得する ……542
307 サーフェイスを画像ファイルにする ……543
308 デバッグ時の”ウォッチ”でインターフェイスのメンバを表示するには？ ……545
309 Zバッファを作成しただけなのに実行時例外エラーがでる ……546
310 シェーダーで原因不明の実行時例外エラーがでる ……547
311 シェーダーでテクスチャ、マテリアル等の変更が反映されない ……548
312 メッシュがカメラに近くなると“くり抜かれる”……548
313 重なっているポリゴンに汚い“しましま”が入る ……549

第2部　DirectSoundの極意

第5章　WAVファイルの極意

5-1　WAVファイルを知る

314 WAVファイルとは？ ……554
315 WAVファイルの構造 ……554

316　WAVファイルの種類 …… 555
317　RIFFとは？ …… 555
318　PCMとは？ …… 557
319　リニアPCMとは？ …… 558
320　標本化の原理 …… 558
321　量子化の原理 …… 560
322　符号化の原理 …… 561
323　サウンドフォーマット構造体を把握する …… 562

第6章　初期化の極意

6-1 DirectSoundの初期化

324　DirectSoundオブジェクトを作成する …… 566
325　協調レベルを設定する …… 566

6-2 WAVデータ読み込み

326　WAVファイルを開く …… 567
327　WAVファイル内を走査してチャンクを見つける …… 568
328　WAVファイルかどうかを判別する …… 568
329　リニアPCMかどうかを判別する …… 569
330　FOURCCとは？ …… 570
331　セカンダリーバッファを作成する …… 571
332　セカンダリバッファにWAVフォーマットを格納する …… 573
333　セカンダリーバッファに波形データを書き込む …… 574
334　WAVファイル読み込み全コード …… 575

第7章　再生の極意

7-1 基本再生

335　セカンダリーバッファを再生する …… 578
336　セカンダリーバッファを停止する …… 578
337　いま再生中かどうか判断する …… 579
338　静的バッファによる基本的な再生を行う …… 580
339　サウンドをパンさせる …… 582
340　周波数を変更する …… 584
341　ボリュームを変更する …… 585
342　サウンドカードのローカルメモリってあるの？ …… 586
343　バックグラウンドでも鳴るようにする …… 587

7-2 ストリーミング再生

344　ストリーミング再生とは？ …… 588
345　なぜストリーミングするの？ …… 588
346　ストリーミング再生はあくまでアプリケーション実装 …… 589
347　ストリーミングの仕組み …… 589
348　ラップアラウンドとは？ …… 593

349 再生カーソルと書き込みカーソルって何？ ……593
350 なぜ2つのポインターが帰ってくるの？ ……594
351 環状バッファのコツ ……595
352 書き込みカーソルは混乱のもと！ ……596
353 WAVデータ上の“読み込みカーソル”が必要！ ……596
354 メモリー括方式とファイル逐次方式 ……597
355 ストリーミング再生を行う（メモリー括保存バージョン）……598
356 完全なストリーミング再生を行う（ファイル逐次読み込みバージョン）……600

7-3 エフェクト再生

＜エフェクト全般＞

357 エフェクトとは？ ……601
358 エフェクトを分類してみる ……602
359 ディレイ系とは？ ……603
360 歪み（ひずみ）系とは？ ……604
361 うねり系とは？ ……604
362 フィルター系とは？ ……604
363 セカンダリバッファにエフェクトをかける ……605
364 セカンダリバッファのエフェクトを解除する ……606
365 複数のエフェクトを重ねる ……607

＜コーラス＞

366 コーラスとは？ ……609
367 コーラスをかける ……609
368 コーラス効果を調整する ……610

＜コンプレッション＞

369 コンプレッションとは？ ……612
370 コンプレッションをかける ……614
371 コンプレッション効果を調整する ……615

＜ディストーション＞

372 ディストーションとは？ ……618
373 ディストーションをかける ……620
374 ディストーション効果を調整する ……621

＜エコー＞

375 エコーとは？ ……622
376 エコーをかける ……623
377 エコー効果を調整する ……624

＜フランジャー＞

378 フランジャーとは？ ……626
379 フランジャーをかける ……626
380 フランジャー効果を調整する ……627

＜ガーグル＞

381 ガーグルとは？ ……629
382 ガーグルをかける ……629
383 ガーグル効果を調整する ……630

<ウェーブス・リバーブ>

384　ウェーブス・リバーブとは？ ……632
385　ウェーブス・リバーブをかける ……632
386　ウェーブス・リバーブ効果を調整する ……633

<I3DL2リバーブ>

387　I3DL2リバーブとは？ ……635
388　I3DL2リバーブをかける ……635
389　I3DL2リバーブ効果を調整する ……637

<パラメトリック・イコライザー>

390　パラメトリック・イコライザーとは？ ……650
391　パラメトリック・イコライザーをかける ……650
392　パラメトリック・イコライザー効果を調整する ……652

7-4　3D再生

393　3Dサウンドとは？ ……653
394　HRTFとは？ ……654
395　セカンダリバッファに3D効果をかける ……654
396　音源を空間的に移動する ……657
397　音源を空間的に回転させる ……658
398　音源の最小・最大距離を設定する ……659
399　音源の速度を設定する ……660
400　リスナーとは？ ……661
401　リスナーを空間的に移動する ……661
402　リスナーを空間的に回転させる ……662
403　リスナーの速度を設定する ……662
404　リスナーにドップラー効果をかける ……663

7-5　マルチチャンネル再生

405　マルチチャンネルとは？ ……664
406　マルチチャンネルのハード環境を整える ……665
407　マルチチャンネルWAVファイルをマルチ再生する ……665
408　マルチチャンネルWAVファイルをモノラルにダウンミックスする ……666
409　マルチチャンネルWAVファイルをステレオにダウンミックスする ……669
410　マルチチャンネルWAVファイルの特定チャンネルを再生する ……670
411　4つのモノラルWAVファイルをマルチチャンネル再生する ……671
412　6つのモノラルWAVファイルを5.1チャンネルで鳴らす ……671
413　WAVファイルがマルチチャンネルかどうか調べる ……672
414　マルチチャンネルのセカンダリバッファを作成する ……673
415　特定チャンネルのセカンダリーバッファを作成する ……673
416　WAVE_FORMAT_EXTENSIBLE（およびWAVEFORMATEX）構造体 ……675
417　WAVデータフォーマットの詳細 ……677

第8章　録音の極意

8-1　サウンドのキャプチャー

418　サウンドのキャプチャーとは？ ……682
419　今スピーカーから出ている音を録音する ……682
420　静的バッファで録音（簡単版）……683
421　ストリームバッファで録音（完全版）……684
422　キャプチャーデバイスってサウンドカードと違うの？ ……685
423　全二重、半二重とは？ ……685
424　キャプチャーデバイスを作成する ……686
425　キャプチャーバッファを作成する ……687
426　ソース側とバッファ側のフォーマットは同じである必要はない ……688
427　キャプチャーを開始する ……689
428　キャプチャーを停止する ……690
429　RIFFにおけるチャンクおよびID等のサイズ……691
430　キャプチャーしたサウンドをWAVファイルにする ……692

第9章　サウンドカードの調査の極意

9-1　サウンドカード

431　利用可能なサウンドカードを列挙する ……696
432　サウンドカードの再生能力を調べる ……697
433　サウンドカードのキャプチャー能力を調べる ……698

第3部　DirectInputの極意

第10章　入力（ユーザー→デバイス）の極意

10-1　初期化

434　キーボード初期化の流れ ……704
435　マウス初期化の流れ ……704
436　ゲームコントローラー初期化の流れ ……704
437　DirectInputオブジェクトを作成する ……705
438　キーボードのDirectInputデバイスオブジェクトを作成する ……706
439　マウスのDirectInputデバイスオブジェクトを作成する ……706
440　ゲームコントローラーのDirectInputデバイスオブジェクトを作成する ……707
441　ゲームコントローラーを列挙する ……708
442　協調レベルとは？ ……710
443　協調レベルの設定 ……710
444　デバイスの取得とは？ ……711
445　デバイスを取得する ……711
446　キーボード初期化 ……712
447　マウス初期化 ……712
448　ゲームコントローラー初期化 ……713

10-2　入力データ取得

449　押下されているキー（キーボード）を取得する……715
450　マウスの動きを取得する……716
451　マウスボタンの押下を取得する……718
452　ゲームコントローラーのユーザー操作を取得する……719
453　アクセス権の消失に対処する……720

第11章　フォースフィードバック（デバイス→ユーザー）の極意

11-1　FFBの基本と共通事項

454　フォースフィードバックとは？……722
455　バイブレーションとフォースフィードバック……722
456　マグニチュードとは？……723
457　DirectInput時間の単位は？……723
458　ディレクションとは？……724
459　エンベロープとは？……725
460　サチュレイションとは？……725
461　コウエフィシェントとは？……727
462　なぜオートセンターを無効にするの？……728
463　フォースの種類について……728

11-2　能動的フォース（ユーザー入力非感応タイプ）

<コンスタント（一定）フォース>

464　コンスタントフォースとは？……729
465　コンスタントフォースを初期化する……729
466　コンスタントフォースをかける……731

<ランプ（傾斜）フォース>

467　ランプフォースとは？……732
468　ランプフォースを初期化する……732
469　ランプフォースをかける……734

<ピリオディック（周期）フォース>

470　ピリオディックフォースとは？……735
471　ピリオディックフォースのパターンについて……735
472　スクエア（矩形波）パターンの初期化……736
473　サイン（正弦波）パターンの初期化……737
474　トライアングル（三角波）パターンの初期化……739
475　ソウトゥース（ノコギリ刃）パターンの初期化……740
476　ピリオディックフォースをかける……741

11-3　受動的フォース（ユーザー入力感応タイプ）

<コンディション（条件）フォース>

477　コンディションフォースとは？……743
478　コンディションフォースの種類について……743
479　スプリング（ばね）の初期化……744
480　フリクション（摩擦）の初期化……745

481　ダンパー（緩衝器）の初期化 ……747
482　イナーシャ（慣性）の初期化 ……748
483　コンディションフォースを有効にする ……749

11-4　FFB機器の調査

484　FFBコントローラーを検出する ……751
485　FFBコントローラーの能力を取得する ……752

第4部　DirectSetupの極意

第12章　DirectSetupの極意

12-1　DirectSetup

486　DirectSetupとは？ ……756
487　再頒布とは？　Redistとは？ ……757
488　EULAとは？ ……758
489　DirectXランタイムバージョン番号について ……759
490　D3DX系のdllバージョン番号について ……760
491　D3DX系のdllバージョン番号を知る方法 ……761
492　そもそもD3DXCheckVersion関数が動かないのでは？ ……762
493　D3DX_VERSION定数でのチェックは危険！ ……763
494　再頒布処理に必要なファイル ……764
495　“DirectXアンインストール不可能” 問題について ……765
496　EULAを表示する ……766
497　結局DirectSetupは必要なの？ ……767
498　留意事項まとめ ……767
499　自分のゲームにDirectXセットアップを実装する ……768

500　ゲームクリエイターとして ……771

索引 ……773
添付CD-ROMについて ……782

DirectX

500 Tips to Use DirectX Better!

序章

001〜006

準備

001 DirectXによるゲーム開発

●DirectXはハイクオリティゲームの開発に必要なAPI

初級

Microsoft DirectX（以降DirectXと表記）は、ウィンドウズプラットフォーム上での高速でハイクオリティなゲームの実行に必要なコンポーネントであり、開発者にとってはハイクオリティゲームの開発に必要なAPI（関数ライブラリのようなもの）でもあります。

DirectXは、ウィンドウズPCのみならずXBOXというゲームコンソール（コンシューマーマシンとも呼ばれる家庭用ゲーム機のことです）においても、そこでのゲームの実行及び開発に必要なものとなっています。

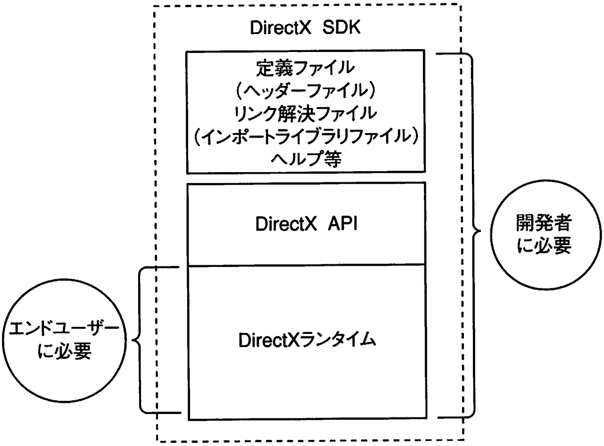
ゲームを遊ぶ側の人であるエンドユーザーにとって、DirectXはゲームの実行にだけ必要なものです。他方、ゲームを作る側の人であるゲーム開発者にとってDirectXは実行と開発の両方の局面で必要なものとなります。エンドユーザーの興味はゲームそのものであってDirectXではないわけですから、DirectXの存在を意識しない人も少なくないでしょう。

ところがゲーム開発者は、純粋にゲームそのものを構築することは当然ですが、それと同時に、足回りであるDirectXを操ることができなくてはなりません。本書はもちろんDirectXによるゲーム開発を行う開発者のための書籍であり、開発者がDirectXを自由に操るためのリファレンス、サンプルコードとその解説、ティップス集です。本書ではDirectX SDKとC++という最強の組み合わせにおけるDirectXコーディングを取り扱います。

エンドユーザーにとって必要なのはDirectXそのものと表現したことに疑問を持った読者もいるかもしれません。他に何かあるかのような言い回しですね。一口にDirectXと言ってもエンドユーザーと開発者ではその範囲が異なります。技術的に言うとDirectXそのものはDirectXランタイムと呼ばれます。エンドユーザーはDirectXランタイムさえあればゲームを動作させることができるので、事足ります。

しかし開発する場合は、DirectXランタイムにアクセスしてそれを操るための何らかのプログラムインターフェイスが当然必要です。それがDirectX APIと呼ばれるAPI（Application Programming Interface）です。そしてさらに実務上、APIを自身のアプリケーション内で使用するためには、任意の言語（本書ではC++）から利用できるための定義（ヘッダーファイル）やリンク（インポートライブラリファイル）が必要になります。それらを一まとめにした開発キットがSDK（Software Development Kit）であり、開発者にはSDKが必要です。

次のような図にすると分かりやすいでしょうか。



厳密に言うと本書（や他の多くの類書）は“DirectXの解説書”ではなく、“DirectX APIの解説書”あるいは“DirectX SDKの解説書”と言ったほうが正確です。DirectXそのものの解説は、マイクロソフトがDirectXランタイムの内部を公開（非常に考えにくいことです）しない限り書けないのですから。

この最も基本的で根本的な分類はプロのプログラマー（エキスパートという意味ではなく、プログラムを職業にしている人）でも曖昧な人は少なくないと思います。

DirectXは高速なマルチメディア処理には欠かせないものです。そしてDirectXに対抗できるようなコンポーネントは存在しませんし、今後そのようなものが登場するとは思えません。ただし特定のコンポーネントごとになら匹敵しているものもあります（ありました）。Direct3Dに対するOpenGL、DirectSound3Dに対するA3D、また、ドライバーレベルのライブラリとしてDirectSoundに対するASIOというドライバーがあったりしますが、いずれも高速であるが機能が乏しい、機能が匹敵していても速度が遅いといったように片手落ちです。また、プログラムからのアクセス性にいたるとDirectXに大きく劣ります（OpenGLは別です）。ただ、通信関係を掌るDirectPlayに関してはWinSockのほうが個人的には優れていると思いますが、WinSockはウィンドウズ組み込みのコンポーネントであるので次元が異なるような気はします。これほど高機能で巨大なコンポーネントであるDirectXは、さらに進化し続けていますので、もはやそれに対抗することは現実的ではないように感じます。

今後DirectXが名前を変えることは可能性としては考えられますが、今までの経緯を鑑ると、前バージョンに少しずつ改良を重ねながら進化をしてきたのがDirectXですから、いきなり別のものになるなどということはないと思います。したがって、その時点で勉強したことはその後のウィンドウズプラットフォーム上でのゲーム開発に必ず役に立つと思います。

筆者はかつてDirectXに依存したコーディングにどっぷり漬かっていてる状況の中で、「これでいいのだろうか？」、などと不安になりながらコーディングしていましたが、今はその不安はありません。DirectXがもしこの世から無くなったとしてもDirectXを操作するために培ったビジュアル、オーディオ、通

信等の普遍的なコーディングエッセンスは永久に自分のものになりますし、DirectXAPIに直結したコード部分は下位ラッパークラスとして分離しておけば未知のドライバーレベルAPIに対応できるわけで、大した問題ではないからです。

何れにしても我々は、現在最も最強で将来性のあるDirectXをいかに上手く利用・活用するかということを考えるのが賢い行動ではないでしょうか。

One Point

DirectX（SDK）とC++は、最強の組み合わせであり、かつ、その地位が揺らぐことはないでしょう。身につけたスキルは確実な財産となります。

002 必要なものについて

●プログラムを「実行するためのもの」と「作るためのもの」が必要　初級

本書のスタイルが基本的にリファレンスであるので、リファレンスとして本書を開いたのであれば、読者は既にDirectXをコーディングできる環境を整備していると考えています。つまり、何らかのDirectXプログラムを作成していて、その中での部分的な問題解決なり部分的な改良をするために各項目を探すという使用方法を想定しています。そうであればここでの解説は不要になります。

他方で、上手く動かない時の環境チェックが目的であったり、さらには、リファレンス目的ではなく本書を購入される場合も可能性としてはあると思います。そのような読者のために、ハードウェア及びソフトウェアとして用意しなければならないものを記しておくことにします。

ただ、本書は文字通りDirectX、すなわちビジュアル（Direct3D）、オーディオ(DirectSound)、操作入力（DirectInput）という全ての要素が範囲であるので、すべてのコンポーネントにとっての必要なものとなると、通常の解説書よりは多くのものが必要になる可能性があります。

とは言うものの、現在のPCは最初から全て揃っているのが通常なので心配する必要は余りないのかもしれません。ビデオカードが入っていないPCは有り得ないですし、サウンドハードウェアが入っていないPCもまずないでしょう。

もちろん、およそゲームに使用され得るデバイス全てに対応しているDirectXですから、DirectX全般を守備範囲とする本書で、解説している全サンプルの動作確認ができるような環境は少ないとは思います。しかし、先述したようにリファレンスであるので、例えばDirectInputにおけるフォースフィードバックの項目を開いた読者は、既にFFBホイールやFFBジョイスティック（FFB=Force FeedBack）を持っている可能性が高いのではないかと思います。また、マルチチャンネルの項目を開いた読者であれば、3スピーカー以上のマルチチャンネルスピーカー環境を整えている人ではないかと思います。

■ハードウェア

最低限必要なものはWidowsパソコンとモ

ニターです。

次にDirectXの各コンポーネントごとに必要なものを述べていきます。

●Direct3Dに最低限度必要なハードウェア

Direct3D9に対応したビデオカード（グラフィックボード）。

お使いのビデオカードがDirect3D8世代以前のものでも、最新のデバイスドライバーを入手し、インストールすることによりDirect3D9に対応できる場合があります。デバイスドライバーの入手はビデオカードの製造元から、インターネットを通したダウンロードにより入手するのが一般的でしょう。

●DirectSoundとDirectMusicに最低限度必要なハードウェア

DirectSound8に対応したサウンドカード（あるいはオーディオカード）。

DirectSoundはDirectX8インターフェイス（DirectX9より古いインターフェイス）であるので、サウンドカードがかなり古くても（5、6年前のものでも）対応していると思われます。

●DirectInputに最低限度必要なハードウェア

（あらためて記述することではないとは思いますが）、キーボード、マウスは必須です。

あと、フォースフィードバックを体感するには、フォースフィードバック機能のあるコントローラーが必要です。スティックタイプ（ジョイスティック）、ハンドルタイプ（ホイールとも呼ばれます）、パッドタイプ（ゲームパッド）がありますが、単に振動を体感するだけであれば、それら何れかがあれば足ります。しかし、本当のフォースフィードバックを体感するのであれば、ハンドルまたはスティックが必要です。

●DirectSetupに必要なハードウェア

DirectSetupに独自に必要となるハードウェアは特にありません。

■ソフトウェア

OS：WindowsXP

●SDK

正確にはDirectX9.0SDK　Update（December2005）です。SDKは読者において適宜入手してください。マイクロソフトのサイトから無償でダウンロードできます。ネット環境が無い人のためにCD-ROM販売も行っているようです。その場合マイクロソフトに問い合わせてください。

なお、2006年3月現在で、最新のSDKはFebruary2006ですが、これをインストールするとコントロールパネルのDirectXプロパティが消えるので、読者がもしFebruary2006以降のSDKをインストールして、コントロールパネルのDirectXが消えているのなら、December2005　SDKをインストールすれば、Directアイコンは元に戻ります。

●コンパイラー

次のうちの何れか1つ。

VisualStudio.NET2005
VisualStudio.NET2003
VisualStudio.NET2002
VisualC++.NET2003
VisualC++.NET2002

One Point

必須のもの以外、例えばDirectSoundにおけるマルチスピーカー、DirectInputにおけるコントローラーは必須ではなく、それらのコーディングを行う人のみに必要です。

003 コンパイラー(Visual C++)とSDKの設定

●コンパイラーからSDKが見えるようにする

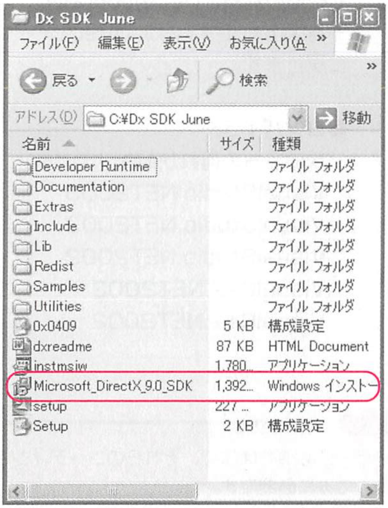
初級

●SDKのインストール

すでに、このバージョンのSDKをインストールしている方はもちろんインストールの必要はありませので、次のステップ「コンパイラー（VisualC++）の設定」に進んでください。

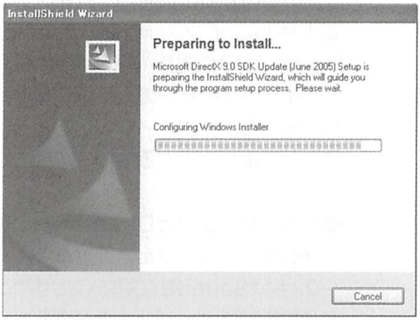
SDKが全くインストールされていないか、または古いSDKがインストールされている場合は、SDKフォルダ内のSetup.exeをダブルクリックして起動すれば、自動的にインストールが進行します。

インストールするには、まずSDKフォルダ内のsetup.exeをダブルクリックして起動します。なお、図中ではSDKのバージョンがJune2005になっていますが、December2005でも全く同じなので、December2005と思って（読み替えて）ください。


Dx SDK June
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
戻る
検索
アドレス(D)
C:¥Dx SDK June
移動

名前	サイズ	種類
Developer Runtime		ファイル フォルダ
Documentation		ファイル フォルダ
Extras		ファイル フォルダ
Include		ファイル フォルダ
Lib		ファイル フォルダ
Redist		ファイル フォルダ
Samples		ファイル フォルダ
Utilities		ファイル フォルダ
0x0409	5 KB	構成設定
dxreadme	87 KB	HTML Document
instmsiw	1,780...	アプリケーション
Microsoft_DirectX_9.0_SDK	1,392...	Windows インストー
setup	227 ...	アプリケーション
Setup	2 KB	構成設定

setup.exeを起動すると次のようにインストールウィザードが開始されます。

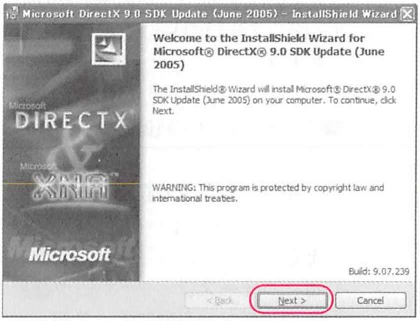

InstallShield Wizard
Cancel

Preparing to Install...

Microsoft DirectX 9.0 SDK Update (June 2005) Setup is preparing the InstallShield Wizard, which will guide you through the program setup process. Please wait.

Configuring Windows Installer

Welcome to the ～の画面になるので、「Next」をクリックします。


Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShield Wizard
Microsoft
DIRECTX
Microsoft
Microsoft
< Back
Next >
Cancel

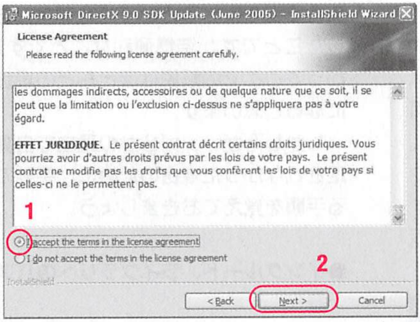
Welcome to the InstallShield Wizard for Microsoft® DirectX® 9.0 SDK Update (June 2005)

The InstallShield® Wizard will install Microsoft® DirectX® 9.0 SDK Update (June 2005) on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

Build: 9.07.239

「accept the terms...」（使用条件に同意）をチェックしてから、「Next」ボタンをクリックします。



Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

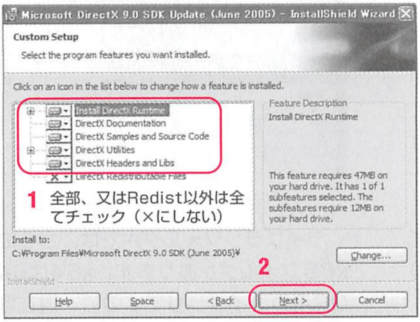
EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

● I accept the terms in the license agreement

○ I do not accept the terms in the license agreement

インストールするファイルを選択するペイン（領域）が表示されます。
基本的にRedist（一番下）以外は全てインストールします。なおRedistをインストールしてももちろん問題ありません。

Redist以外が「×（バツ）」になっていないことを確認してから、「Next」ボタンをクリックします。



Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShield Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- Install DirectX Runtime
- DirectX Documentation
- DirectX Samples and Source Code
- DirectX Utilities
- DirectX Headers and Libs
- ✕ DirectX Redistributable Files

Feature Description
Install DirectX Runtime

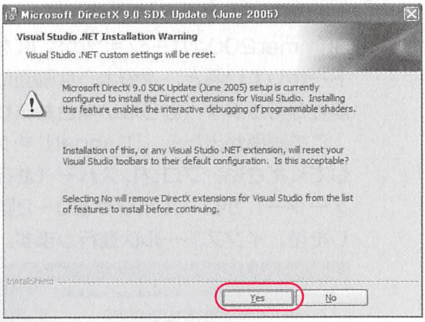
This feature requires 47MB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 12MB on your hard drive.

Install to:
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥

1 全部、又はRedist以外は全てチェック（×にしない）

「Yes」をクリックするとSDKのインストーラーが、コンパイラー（Visual C++）におけるDirect3DエクステンションとDirect3Dシェーダーの設定を行います。なお、コンパイラーの個人的な設定がリセットされてしまうことがありますが、それは再度設定するようにすればいいことです（エディターのフォント設定やショートカットキーはリセットされません）。

「No」をクリックするとそれらの設定は行われませんので、シェーダーのデバッグが困難になります。「Yes」をクリックするようにしてください。



Microsoft DirectX 9.0 SDK Update (June 2005)

Visual Studio .NET Installation Warning

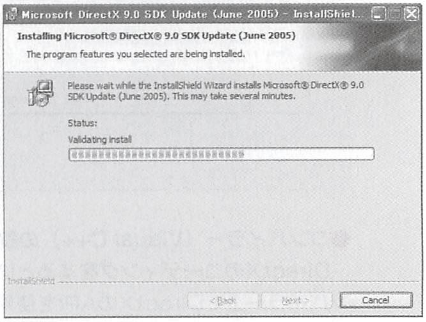
Visual Studio .NET custom settings will be reset.

Microsoft DirectX 9.0 SDK Update (June 2005) setup is currently configured to install the DirectX extensions for Visual Studio. Installing this feature enables the interactive debugging of programmable shaders.

Installation of this, or any Visual Studio .NET extension, will reset your Visual Studio toolbars to their default configuration. Is this acceptable?

Selecting No will remove DirectX extensions for Visual Studio from the list of features to install before continuing.

インストールが始まります。



Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShiel...

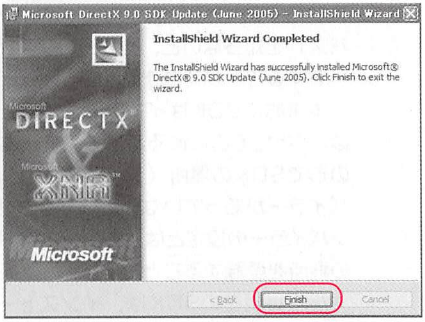
Installing Microsoft® DirectX® 9.0 SDK Update (June 2005)

The program features you selected are being installed.

Please wait while the InstallShield Wizard installs Microsoft® DirectX® 9.0 SDK Update (June 2005). This may take several minutes.

Status:
Validating install

インストールが始まれば、あとはインストールの完了画面を待つだけです。次の画面になったらインストール完了です。「Finish」をクリックしてインストールウィザードを終了しましょう。



Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShield Wizard

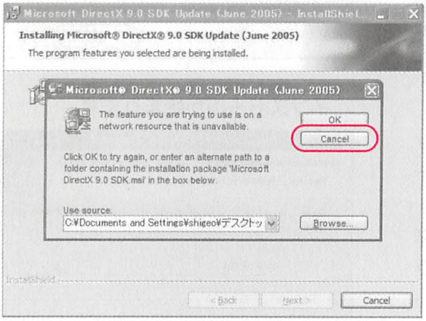
InstallShield Wizard Completed

The InstallShield Wizard has successfully installed Microsoft® DirectX® 9.0 SDK Update (June 2005). Click Finish to exit the wizard.

●もしもこの画面になってしまったら

以前のSummer2003バージョンやSummer2004バージョンのSDKをインストールしていると、次のような画面になる可能性があります（ならない場合もあります）。

この画面が出たら、「Cancel」をクリックしてください。プログレスバー（進行を表すメーター）がロールバックを0～2回繰り返した後、インストールは進行します。



Microsoft DirectX 9.0 SDK Update (June 2005) - InstallShiel...

Installing Microsoft® DirectX® 9.0 SDK Update (June 2005)

The program features you selected are being installed.

Microsoft® DirectX® 9.0 SDK Update (June 2005)

The feature you are trying to use is on a network resource that is unavailable.

Click OK to try again, or enter an alternate path to a folder containing the installation package 'Microsoft DirectX 9.0 SDK.msi' in the box below.

Use source:
C:¥Documents and Settings¥shigeo¥デスクトッ

OK
Cancel
Browse...
InstallShield
< Back
Next >
Cancel

●コンパイラー（Visual C++）の設定

DirectXのコーディングをするというはコンパイラー上でDirectXのAPIを使用することですから、DirectXのAPIをコールできなければなりません。DirectXのAPIをコール、すなわちAPIにアクセスするためには、コンパイラーがAPIの定義（ヘッダーファイル）、APIの参照解決（ライブラリファイル）を読み込むことができなくてはなりません。コンパイラーがSDKの場所（ディレクトリ、パス）を知らないと、それらのファイルをロードすることができません。

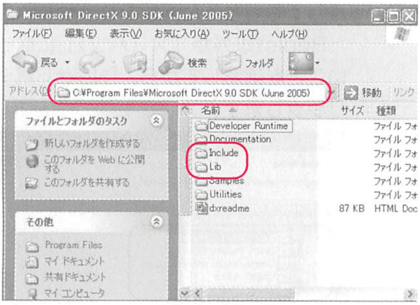
本来的にSDKはインストールしただけでは、“ただそこにあるだけ”です。なんらかの形でSDKの場所（ディレクトリ）をコンパイラーが知っていなければなりません。コンパイラーの設定とは、コンパイラーにSDKの場所を登録することです。

ただし現在のSDKは、インストーラーが自動的にコンパイラー側の設定を行ってくれます。つまり、SDKをインストールさえすれば、それだけでコンパイラーからSDKを使えるようになるということです（ちょっとしたことですが結構便利なことです）。したがってここでの解説はほとんどの場合、不要になると思います。

しかし万が一、何らかの理由で自動的に設定されなかった場合のために、手動で設定する手順を覚えておきましょう。

●インクルード、ライブラリへのパス手動登録

コンパイラーがDirectXプログラムをビルドするために必要なファイルは、SDKをインストールした場所（図3-10のフォルダ）にあるIncludeフォルダとLibフォルダ内にあるファイルです。

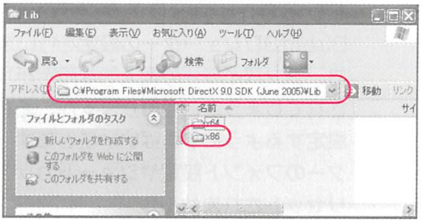


Microsoft DirectX 9.0 SDK (June 2005)

アドレス(D) C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)

ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
ヘルプ(H)
戻る
検索
フォルダ
移動
リンク
ファイルとフォルダのタスク
新しいフォルダを作成する
このフォルダを Web に公開する
このフォルダを共有する
その他
Program Files
マイ ドキュメント
共有ドキュメント
マイ コンピュータ
名前
サイズ
種類
Developer Runtime
Documentation
Include
Lib
Samples
Utilities
dxreadme
87 KB
HTML Doc
ファイル フォ

ライブラリフォルダ内では、さらにx64とx86フォルダがあります。x64は64ビットOS用のライブラリですので、現在ほとんどの環境ではx86フォルダ内のファイルを使用します。



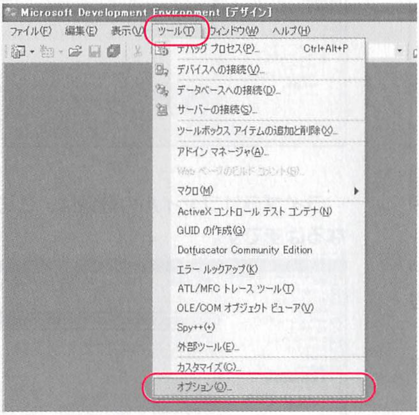
Lib

アドレス(D) C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Lib

ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
ヘルプ(H)
戻る
検索
フォルダ
移動
リンク
ファイルとフォルダのタスク
新しいフォルダを作成する
このフォルダを Web に公開する
このフォルダを共有する
名前
x64
x86

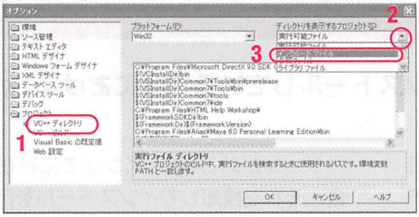
コンパイラーにインクルードファイルとライブラリファイルの場所を登録するには、メ

インメニューの「ツール」→「オプション」をまずクリックします。


Microsoft Development Environment [デザイン]
ファイル(F)
編集(E)
表示(V)
ツール(T)
ウィンドウ(W)
ヘルプ(H)
デバッグ プロセス(P)... Ctrl+Alt+P
デバイスへの接続(V)...
データベースへの接続(D)...
サーバーの接続(S)...
ツールボックス アイテムの追加と削除(X)...
アドイン マネージャ(A)...
Web ページのビルド コメント(B)...
マクロ(M)
ActiveX コントロール テスト コンテナ(N)
GUID の作成(G)
Dotfuscator Community Edition
エラー ルックアップ(K)
ATL/MFC トレース ツール(T)
OLE/COM オブジェクト ビューア(V)
Spy++(+)
外部ツール(E)...
カスタマイズ(C)...
オプション(O)...

次のようなオプション・ダイアログが表示されます。

1. 左ペイン（左側の領域、窓）の一番下、「プロジェクト」をクリック、さらに「VC++ディレクトリ」をクリックすると、右ペインが図のようになります。
2. ダイアログ右上の「ディレクトリを表示するプロジェクト」コンボボックスを展開し、
3. 「インクルードファイル」に合わせます。


オプション
2
3
1
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
プラットフォーム(P):
ディレクトリを表示するプロジェクト(S):
ライブラリ ファイル
OK
キャンセル
ヘルプ

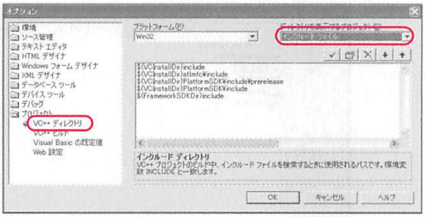
Win32

実行可能ファイル

C:¥Program Files¥Microsoft DirectX 9.0 SDK (
$(VCInstallDir)bin
$(VSInstallDir)Common7¥Tools¥bin¥prerelease
$(VSInstallDir)Common7¥Tools¥bin
$(VSInstallDir)Common7¥tools
$(VSInstallDir)Common7¥ide
C:¥Program Files¥HTML Help Workshop¥
$(FrameworkSDKDir)bin
$(FrameworkDir)$(FrameworkVersion)
C:¥Program Files¥Alias¥Maya 6.0 Personal Learning Edition¥bin

実行ファイル ディレクトリ
VC++ プロジェクトのビルド中、実行ファイルを検索するときに使用されるパスです。環境変数 PATH と一致します。

すると画面が次のようになります。SDKのディレクトリは、まだ未登録の状態です。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
プラットフォーム(P):
ディレクトリを表示するプロジェクト(S):
OK
キャンセル
ヘルプ

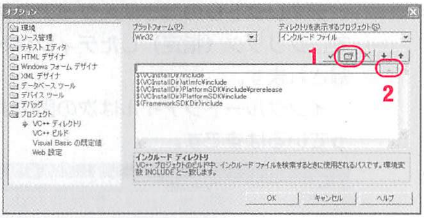
Win32

インクルード ファイル

$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

1. 「新しい行」ボタンを押します。
2. ディレクトリの置き換えダイアログを出すためのボタンをクリックします。


オプション
1
2
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
プラットフォーム(P):
ディレクトリを表示するプロジェクト(S):
OK
キャンセル
ヘルプ

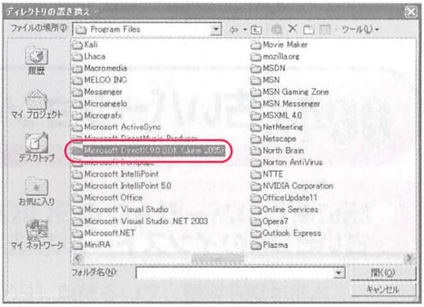
Win32

インクルード ファイル

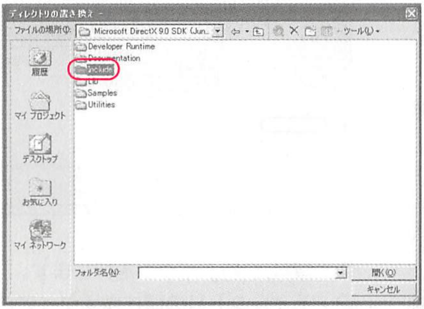
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

次のようなディレクトリの置き換えダイアログボックスが表示されますので、SDKのインクルード及びライブラリファイルのディレクトリを指定します。


ディレクトリの置き換え
ファイルの場所(I):
ツール(L)
履歴
マイ プロジェクト
デスクトップ
お気に入り
マイ ネットワーク
Kali
Lhaca
Macromedia
MELCO INC
Messenger
Microangelo
Micrografx
Microsoft ActiveSync
Microsoft DirectMusic Producer
Microsoft DirectX 9.0 SDK (June 2005)
Microsoft Frontpage
Microsoft IntelliPoint
Microsoft IntelliPoint 5.0
Microsoft Office
Microsoft Visual Studio
Microsoft Visual Studio .NET 2003
Microsoft.NET
MiniRA
Movie Maker
mozilla.org
MSDN
MSN
MSN Gaming Zone
MSN Messenger
MSXML 4.0
NetMeeting
Netscape
North Brain
Norton AntiVirus
NTTE
NVIDIA Corporation
OfficeUpdate11
Online Services
Opera7
Outlook Express
Plazma
フォルダ名(N):
開く(O)
キャンセル

Program Files

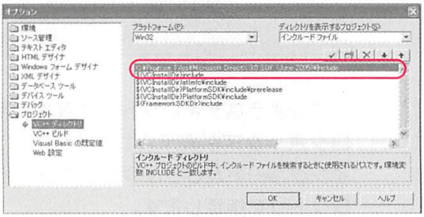

ディレクトリの置き換え
ファイルの場所(I):
ツール(L)
履歴
マイ プロジェクト
デスクトップ
お気に入り
マイ ネットワーク
フォルダ名(N):
開く(O)
キャンセル

ファイルの場所(I): Microsoft DirectX 9.0 SDK (Jun...

Developer Runtime
Documentation
Include
Lib
Samples
Utilities

ディレクトリの置き換えダイアログにおいて、クリック（指定）したディレクトリが登録されます。

インクルードファイルは次の図のようになっているはずです。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

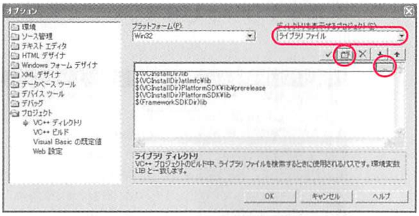
プラットフォーム(P): Win32

ディレクトリを表示するプロジェクト(S): インクルード ファイル

C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

同様にライブラリファイルも登録します。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

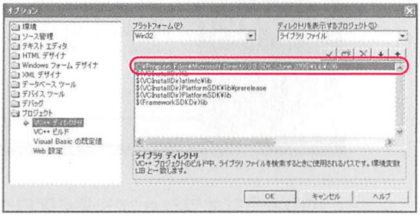
プラットフォーム(P): Win32

ディレクトリを表示するプロジェクト(S): ライブラリ ファイル

$(VCInstallDir)lib
$(VCInstallDir)atlmfc¥lib
$(VCInstallDir)PlatformSDK¥lib¥prerelease
$(VCInstallDir)PlatformSDK¥lib
$(FrameworkSDKDir)lib

ライブラリ ディレクトリ
VC++ プロジェクトのビルド中、ライブラリ ファイルを検索するときに使用されるパスです。環境変数 LIB と一致します。

ライブラリファイルの登録は、このようになるはずです。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

プラットフォーム(P): Win32

ディレクトリを表示するプロジェクト(S): ライブラリ ファイル

C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Lib¥x86
$(VCInstallDir)lib
$(VCInstallDir)atlmfc¥lib
$(VCInstallDir)PlatformSDK¥lib¥prerelease
$(VCInstallDir)PlatformSDK¥lib
$(FrameworkSDKDir)lib

ライブラリ ディレクトリ
VC++ プロジェクトのビルド中、ライブラリ ファイルを検索するときに使用されるパスです。環境変数 LIB と一致します。

以上でSDKの手動登録は完了です。

004

古いバージョンのSDKをインストールする方法

既に新しいバージョンのSDKをインストールしている場合における、古いSDKのインストール方法

初級

本書は現在執筆している時点（2006年3月）での最新バージョンをベースにしていますが、読者がこれを読んでいる時に、より新しいバージョンのSDKがリリースされている可能性があります。最近1年間のリリースペースを考えると、その可能性はかなり高いでしょう。そして読者が既に、その新しいSDKをインストールしているのであれば、以下の設定を行う必要があります。

SDKのインストーラーは、自分よりもバージョンが新しいSDKがすでに存在する場合、それを検知してインストールを行ってくれません。

現在インストールされいてるSDKより、古いバージョンのSDKをインストールしたい場合は、手動でインストールする必要があります。

手動インストールといっても心配はありません。単純に必須ファイル（インクルードファイルとライブラリファイル）をハードディ

スクの適当な場所にコピーするだけです。

ここでは例として、June2005バージョンがすでにインストールされているマシンに、9.0bバージョンを手動インストールする手順を解説しますが、もちろん、この組み合わせ以外でも手順は同じす（9.0bは完全な日本語マニュアルが存在するので、需要は高いと思います）。

手順を箇条書きにすると次のようになります。

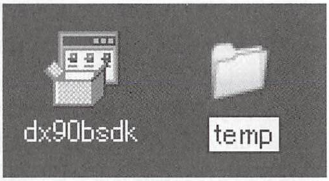
1. SDKパッケージを解凍する（解凍したものがある場合は不要です）。
2. SDKフォルダ内のIncludeフォルダとLibフォルダを適当な場所にコピーする。
3. IncludeフォルダとLibフォルダのディレクトリをコンパイラーに登録する。

では、それぞれの手順を解説します。

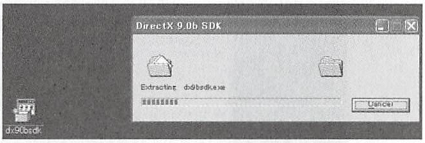
1. SDKパッケージを解凍する。（解凍したものがある場合は不要です。）

次の図中、右にあるのが9.0bのSDKパッケージ（自己解凍ファイル）です。

解凍したファイルを一時的に入れておくフォルダを用意しておきます（デスクトップ上で適当にtempというフォルダを作成しました）。

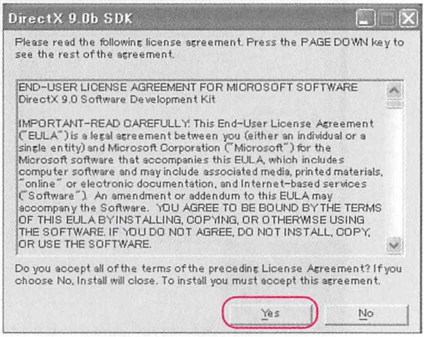


SDKパッケージをダブルクリックして実行すると、次の図のように、まず第1段階の解凍が始まりまります。



Extracting dx9bsdk.exe

第1段階の解凍が終わると、次ような使用条件許諾のダイアログが出るので、「Yes」を押します。



Please read the following license agreement. Press the PAGE DOWN key to see the rest of the agreement.

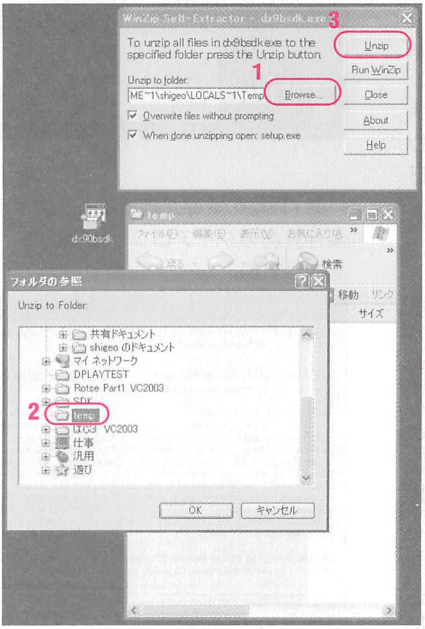
END-USER LICENSE AGREEMENT FOR MICROSOFT SOFTWARE
DirectX 9.0 Software Development Kit

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation ("Microsoft") for the Microsoft software that accompanies this EULA, which includes computer software and may include associated media, printed materials, "online" or electronic documentation, and Internet-based services ("Software"). An amendment or addendum to this EULA may accompany the Software. YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE. IF YOU DO NOT AGREE, DO NOT INSTALL, COPY, OR USE THE SOFTWARE.

Do you accept all of the terms of the preceding License Agreement? If you choose No, Install will close. To install you must accept this agreement.

SDKパッケージに組み込まれているWinZipが起動します。これが最終的な解凍です。この解凍が終了すれば目的は達成されます。古いSDKの設定に必要な作業はインストールではなく解凍されたファイル群から必須ファイルを抜き出すことなのです。

そのままでは、インストーラーに設定されている訳の分からない一時フォルダになっているので、最初に作成しておいたデスクトップ上のTempフォルダに変更します。



WinZip Self-Extractor - dx9bsdk.exe

To unzip all files in dx9bsdk.exe to the specified folder press the Unzip button

Unzip to folder: ME~1\shigeo\LOCALS~1\Temp

☑ Overwrite files without prompting

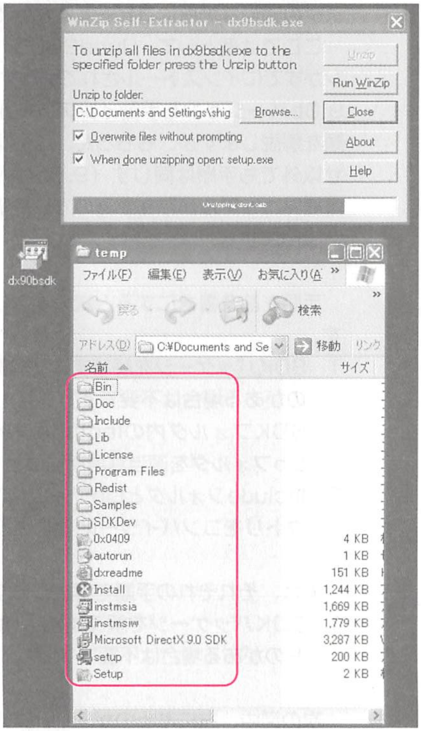
☑ When done unzipping open: setup.exe

フォルダの参照

Unzip to Folder:

3
Unzip
Run WinZip
1
Browse...
Close
About
Help
dx90bsdk
temp
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
検索
移動
リンク
サイズ
共有ドキュメント
shigeo のドキュメント
マイ ネットワーク
DPLAYTEST
Rotse Part1 VC2003
SDK
2
temp
ぱむ3 VC2003
仕事
汎用
遊び
OK
キャンセル

ディレクトリを変更したら、「Unzip」ボタンをクリックします。最終解凍が進行し。ファイル群がtempフォルダに詰め込まれていきます。



WinZip Self-Extractor - dx9bsdk.exe

To unzip all files in dx9bsdk.exe to the specified folder press the Unzip button

Unzip to folder: C:\Documents and Settings\shig

☑ Overwrite files without prompting

☑ When done unzipping open: setup.exe

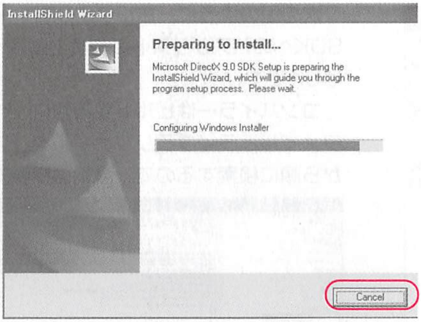
アドレス(D) C:¥Documents and Se

```
名前                        サイズ
Bin
Doc
Include
Lib
License
Program Files
Redist
Samples
SDKDev
0x0409                      4 KB
autorun                     1 KB
dxreadme                  151 KB
Install                 1,244 KB
instmsia                1,669 KB
instmsiw                1,779 KB
Microsoft DirectX 9.0 SDK  3,287 KB
setup                     200 KB
Setup                       2 KB
```

Unzip
Run WinZip
Browse...
Close
About
Help
Unzipping dxnt.cab
dx90bsdk
temp
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
戻る
検索
移動
リンク

解凍が終了した時点で目的は達成されます。なぜなら、Includeフォルダ、Libフォルダが手に入ったからです。

しかしインストーラーは自動的に次の段階（インストール工程）に進もうとします。これ以降の処理は不要なので「Cancel」ボタンを押してインストールを終了してください。ダイアログのすぐに切り替わってしまい、Cancelボタンを押せなかった場合も心配は要りません。どのみち、これ以降の処理を進めたとしても、インストーラーは新しいSDKを検知して、結局インストールを行わずにすぐに終了（Finishの画面になる）します。


InstallShield Wizard
Cancel

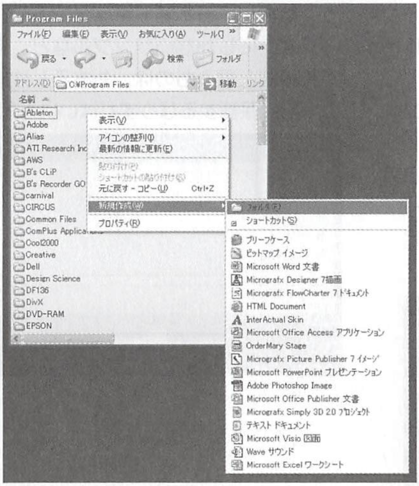
Preparing to Install...

Microsoft DirectX 9.0 SDK Setup is preparing the InstallShield Wizard, which will guide you through the program setup process. Please wait.

Configuring Windows Installer

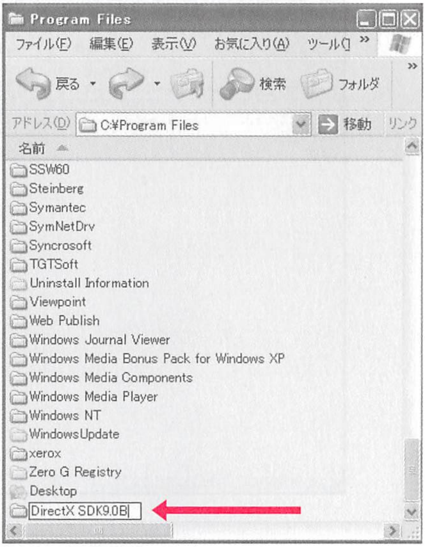
2. SDKフォルダ内のIncludeフォルダと
　Libフォルダを適当な場所にコピーする。

ここまでの段階で必須ファイルは手に入りました。次は、それら必須ファイルを一時的なtempフォルダから正式なフォルダに移動しましょう。

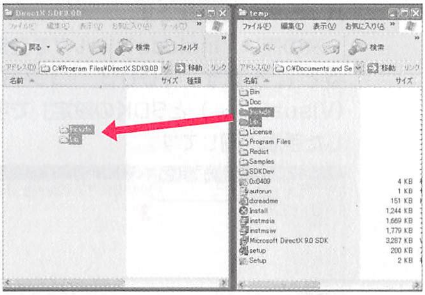

Program Files
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
戻る
検索
フォルダ
アドレス(D)
C:¥Program Files
移動
リンク
名前
Ableton
Adobe
Alias
ATI Research Inc
AWS
B's CLiP
B's Recorder GO
carnival
CIRCUS
Common Files
ComPlus Applications
Cool2000
Creative
Dell
Design Science
DF136
DivX
DVD-RAM
EPSON
表示(V)
アイコンの整列(I)
最新の情報に更新(E)
貼り付け(P)
ショートカットの貼り付け(S)
元に戻す - コピー(U)　Ctrl+Z
新規作成(W)
プロパティ(R)
フォルダ(F)
ショートカット(S)
ブリーフケース
ビットマップ イメージ
Microsoft Word 文書
Micrografx Designer 7描画
Micrografx FlowCharter 7 ドキュメント
HTML Document
InterActual Skin
Microsoft Office Access アプリケーション
OrderMary Stage
Micrografx Picture Publisher 7 イメージ
Microsoft PowerPoint プレゼンテーション
Adobe Photoshop Image
Microsoft Office Publisher 文書
Micrografx Simply 3D 2.0 プロジェクト
テキスト ドキュメント
Microsoft Visio 図面
Wave サウンド
Microsoft Excel ワークシート

正式なフォルダとは、それ以降、コンパイラーが必須ファイルのロード先として参照するフォルダです。

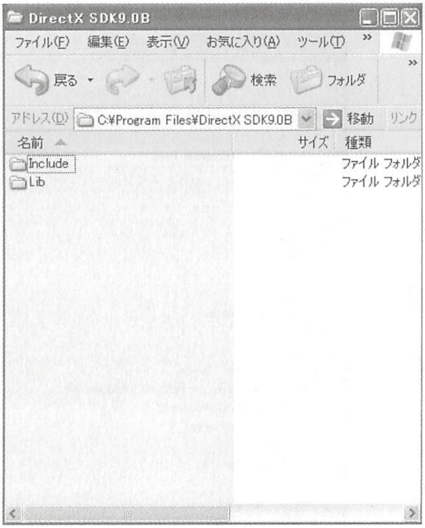
どこでもいいのですが、通常はProgram Filesディレクトリにしましょう。
次の図はProgram　Filesディレクトリ下にDirectX　SDK9.0Bというフォルダを作成しているところです。フォルダ名は任意ですが、バージョンが分かりやすいようにしたほうがいいでしょう。


Program Files
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
戻る
検索
フォルダ
アドレス(D)
C:¥Program Files
移動
リンク
名前
SSW60
Steinberg
Symantec
SymNetDrv
Syncrosoft
TGTSoft
Uninstall Information
Viewpoint
Web Publish
Windows Journal Viewer
Windows Media Bonus Pack for Windows XP
Windows Media Components
Windows Media Player
Windows NT
WindowsUpdate
xerox
Zero G Registry
Desktop
DirectX SDK9.0B

必須ファイルであるInclude、Libフォルダを正式なフォルダに移動します。


DirectX SDK9.0B
temp
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
戻る
検索
フォルダ
C:¥Program Files¥DirectX SDK9.0B
C:¥Documents and Se
移動
名前
サイズ
種類
Include
Lib
Bin
Doc
License
Program Files
Redist
Samples
SDKDev
0x0409
autorun
dxreadme
Install
instmsia
instmsiw
Microsoft DirectX 9.0 SDK
setup
Setup
4 KB
1 KB
151 KB
1,244 KB
1,669 KB
1,779 KB
3,287 KB
200 KB
2 KB

次の図の状態になればファイルの準備は完了です。


DirectX SDK9.0B
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T)
戻る 検索 フォルダ
移動 リンク
名前 サイズ 種類

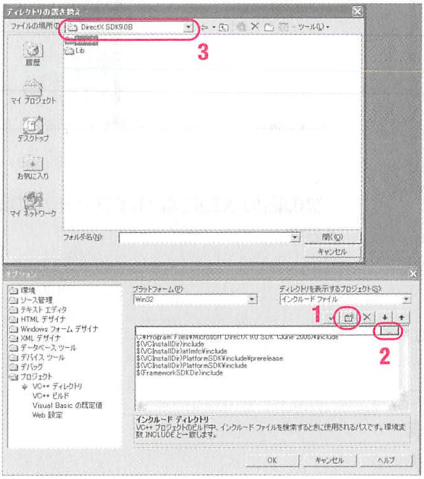
アドレス(D) C:¥Program Files¥DirectX SDK9.0B

Include ファイル フォルダ
Lib ファイル フォルダ

3. IncludeフォルダとLibフォルダのディレクトリをコンパイラーに登録する。

ファイルの準備ができたので、あとはコンパイラー側の登録です。

次の図は、Includeフォルダを登録しているところです。「003　コンパイラー（Visual C++）とSDKの設定」で手動設定した手順と同じです。


ディレクトリの選択
ファイルの場所(I)
履歴
マイ プロジェクト
デスクトップ
お気に入り
マイ ネットワーク
フォルダ名(N)
開く(O)
キャンセル
3
オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
1
2
OK
キャンセル
ヘルプ

ファイルの場所(I): DirectX SDK9.0B

Include
Lib

プラットフォーム(P): Win32

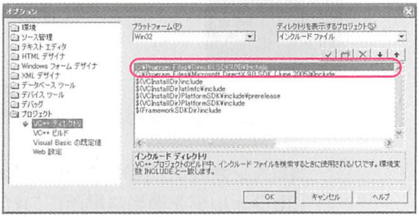
ディレクトリを表示するプロジェクト(S): インクルード ファイル

```
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

次の図のように、古い（9.0Bバージョン）SDKへのパスがリストの一番上になっていればOKです。

コンパイラーはビルドする際に、ヘッダーファイルとライブラリファイルをリストの上から順に検索するので、順番は重要です。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

プラットフォーム(P): Win32

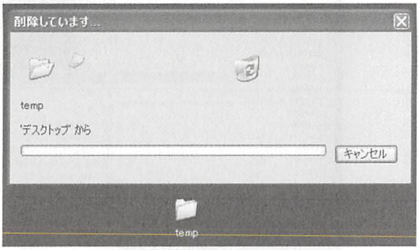
ディレクトリを表示するプロジェクト(S): インクルード ファイル

```
C:¥Program Files¥DirectX SDK9.0B¥include
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

ここまでで、設定は完了です。

なお、一時的なtempフォルダにあるファイル群は、もはや不要ですので削除しましょう。ただし、日本語ドキュメントや、そのバージョンにしかないサンプルなど、自分とって有用だと思うものは、もちろん残すなり別の場所に保管するなりすればいいと思います。



削除しています...

temp

'デスクトップ' から

005 異なるバージョンのSDKを使い分ける方法

古いSDKと新しいSDKを共存させたい場合

初級

基本的にSDKは特定のバージョンのもの1種類だけあればいいのですが、次のような場合には複数のSDKバージョンを使い分けることになります。

●自宅（自分）以外の異なるマシン環境でのプロジェクトを扱わなければならない場合

この状況の具体的な例は、書籍で取り扱うSDKバージョンが自宅のSDKバージョンと異なる場合、又は、ゲーム系スクールに通っている場合は学校と自宅のSDKバージョンが異なる場合が挙げられます。あるいは、友人のプロジェクトをビルドする場合もそうなるでしょう。

●特定のSDKバージョンの使い勝手の関係で複数のSDKを使いたい場合

最新のSDKには日本語訳されたヘルプドキュメントが付いていないことが多く、日本語ヘルプが公開されるのは数ヶ月先か、あるいは公開されない場合もあります。日本語ヘルプのあるバージョンを使ったほうが何かと便利なことはあると思います。

また、特定のSDKにしか付いていないサンプルプログラムがあります。そのサンプルを解読するために、そのバージョンのSDKを使いたい場合もあると思います。

●過去の作品のデバッグ等、また、開発期間が数年単位のプロジェクトを進行している場合

筆者は特に、この状況になります。5年前に開発を始め、3年前に完成したゲームをネットに公開したのですが、最近まで改良やデバッグを行っていました。現在の別プロジェクトはDirectX9用のSDKで行いながら、そのゲームのデバッグ等はDirectX7時代のSDK上で行わなければなりませんでした。

さらに、開発期間が2年くらいあったので、開発中においても新しい別プロジェクトと、そのゲームのプロジェクトは、異なるSDK上でコーディングしなければなりませんでした。

●多くの環境に対応できるゲームを作成するため

コアゲーマーであれば、PC本体はもとよりビデオカードも比較的新しく高機能なものを使用しているので問題はありませんが、一般の人々にプレイしてもらうには、ゲームが要求するマシン環境を低めに設定しなければなりません。より多くの環境でも動作するようなゲームを開発したいと思った場合は、むしろ昔のSDKのほうが都合がいい場合があります。

例えばせっかく作ったゲームを親戚のおばさんに送信し、プレイしてもらおうと思っても、向こうのPC（のビデオカード）がDirectX9に対応していなければでゲームを起動できません。(これはよくあることです)。

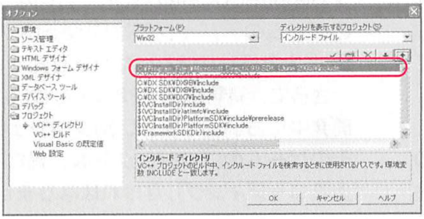
その時もし、そのゲームをDirectX8世代のSDKで作成していたのであれば、おばさんPCで動く確率はかなり高くなり、さらにDirectX7SDK上で開発したのなら、間違いなく動くと言えます。

●コンパイラー側のSDK参照を切り替える

SDKを切り替えること自体は、至極簡単です。コンパイラー側でのSDKへのパスの順番を入れ替えればいいだけです。

次の図は、筆者のコンパイラーにおける実際の状況です。

DirectX7から、最新のSDKまでのパスを登録しています。通常は、最新のJune2005を使用しているので、June2005をリストの最上部に置いています。

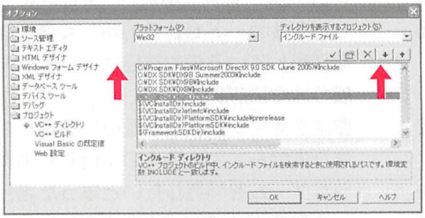

オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

プラットフォーム(P): Win32
ディレクトリを表示するプロジェクト(S): インクルード ファイル

```
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Include
C:¥DX SDK¥DX9C Summer2004¥Include
C:¥DX SDK¥DX9B¥Include
C:¥DX SDK¥DX8¥Include
C:¥DX SDK¥DX7¥Include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

DirectX7で書いた昔のゲーム作品をデバッグする時は、DirectX7SDKのパスを最上部に移動します。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

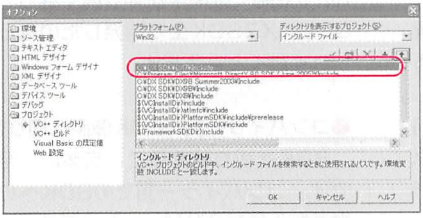
プラットフォーム(P): Win32
ディレクトリを表示するプロジェクト(S): インクルード ファイル

```
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Include
C:¥DX SDK¥DX9B Summer2003¥Include
C:¥DX SDK¥DX9B¥Include
C:¥DX SDK¥DX8¥Include
C:¥DX SDK¥DX7¥Include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

次のような状態にすれば、DirectX7でコーディング、ビルド、実行できます。

先にも述べましたが、コンパイラーはこのリストの上から順番に検索します。そのため、例えばDirectX7とDirectX9のSDKに同じ名前のメソッドがあったとしても、DirectX7のほうのメソッドとして解釈（コンパイル、リンク）されます。

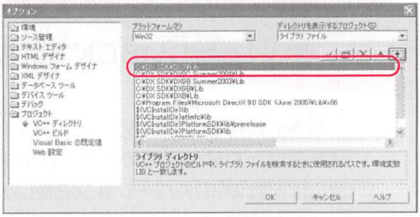

オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

プラットフォーム(P): Win32
ディレクトリを表示するプロジェクト(S): インクルード ファイル

```
C:¥DX SDK¥DX7¥Include
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Include
C:¥DX SDK¥DX9B Summer2003¥Include
C:¥DX SDK¥DX9B¥Include
C:¥DX SDK¥DX8¥Include
$(VCInstallDir)include
$(VCInstallDir)atlmfc¥include
$(VCInstallDir)PlatformSDK¥include¥prerelease
$(VCInstallDir)PlatformSDK¥include
$(FrameworkSDKDir)include
```

インクルード ディレクトリ
VC++ プロジェクトのビルド中、インクルード ファイルを検索するときに使用されるパスです。環境変数 INCLUDE と一致します。

同様に、ライブラリファイルへのパスもDirectX7のものを最上部に移動すれば切り替え完了です。


オプション
環境
ソース管理
テキスト エディタ
HTML デザイナ
Windows フォーム デザイナ
XML デザイナ
データベース ツール
デバイス ツール
デバッグ
プロジェクト
VC++ ディレクトリ
VC++ ビルド
Visual Basic の既定値
Web 設定
OK
キャンセル
ヘルプ

プラットフォーム(P): Win32
ディレクトリを表示するプロジェクト(S): ライブラリ ファイル

```
C:¥DX SDK¥DX7¥Lib
C:¥DX SDK¥DX9C Summer2004¥Lib
C:¥DX SDK¥DX9B Summer2003¥Lib
C:¥DX SDK¥DX9B¥Lib
C:¥DX SDK¥DX8¥Lib
C:¥Program Files¥Microsoft DirectX 9.0 SDK (June 2005)¥Lib¥x86
$(VCInstallDir)lib
$(VCInstallDir)atlmfc¥lib
$(VCInstallDir)PlatformSDK¥lib¥prerelease
$(VCInstallDir)PlatformSDK¥lib
```

ライブラリ ディレクトリ
VC++ プロジェクトのビルド中、ライブラリ ファイルを検索するときに使用されるパスです。環境変数 LIB と一致します。

これだけです。しかし、読者はこれらの図を見て、「このパスの先にあるファイルはどのように配置しているのか」という疑問が浮かぶのではないでしょうか。

複数のSDKをどのように配置するか？

手間がかかるのは、切り替える以前の段階、つまり複数のSDKをハードディスク上にどのように配置するかということだと思います。

Tips004で解説した手順により、それぞれのヘッダーファイルとライブラリファイルを適当なディレクトリのフォルダ内に置きます。その2種類のファイルさえあれば、そのバージョンのSDKは使えます。

それぞれのSDKにそれぞれのランタイムは必要にならないのでしょうか？　なぜ、そのようなごく少量のファイルで大丈夫なのでしょう？

それは、DirectXだからです。変な言い方ですが、次を読んでください。

DirectXはCOMの仕様に沿ったコンポーネントであり、厳格な下位互換性を持っています。下位互換性とは、新しいバージョンさえあれば、それより古いバージョンも必ず使用できるという性質です。DirectXが下位互換性を保つので、過去のAPIは“必ず“使用できます。

DirectXランタイム内には、過去から現在までの全てのAPI（古いAPIも新しいAPIも）に対応するバイナリコードが必ず存在します。実はDirectXは下位互換性の絶対ルールのもと、DirectX2のリリースから今日の

DirectX9までずっと“付け足し”のアップデートがなされているのです。

このような構造により、安心して上位バージョンにアップデートすることができます。上位バージョンにアップデートしたことにより、過去のプログラムが動かなくなるなんていうことは原則的にありません。そもそもそれがDirectXの大きな“売り”でもあるわけですから。したがって逆に言うと、過去のAPIが使えなければDirectXではありませんので（DirectXの定義から外れる）、そのように言い切れるわけです。

古いヘッダーファイルは過去のAPIの型定義であり、古いAPIのコンパイル時に適用されます。そして、古いライブラリファイルは過去のAPIのDLL内（例えばd3d9.dll）での居場所であり、古いAPIのリンク時に適用されます。平たく言えば、ヘッダーとライブラリは、最新のランタイムから古いAPI用のバイナリコードを抜き出してくれる“手掛かり”です。それらさえあればビルドできます。さらに、APIに対応したバイナリコードは必ず存在するので、当然実行もできます。
長くなりましたので、そろそろ実際の手順を解説します。

とは言っても、手順はTips004と同じです。というか、Tips004はこの部分の解説です。ですので、解説することは何もありません。

Tips004の手順により、複数のSDKのヘッダーとライブラリファイルを好きなだけ適用な場所に配置してください。

One Point

SDKは、その時のプロジェクトによってどのようなバージョンにも切り替えることができます。

006 本書におけるフレームワーク（コード全体の構造）の特徴

本書サンプルコードのスタイルはシンプル

初級

●特徴

本書におけるサンプルプロジェクトは次の3つの特徴があります。

1．クラス形式ではない
2．ソースファイルは原則的に1つのみ
3．エラー処理は必要最低限のものだけ

基本的に、ほとんどのサンプルプロジェクトはクラス形式では書きません。クラスベースでコーディングすると、メンテナンスがしやすく、再利用性も高いので。筆者は通常の業務（学習用サンプルを除く）ではクラス形式でしか書きません。本業のゲームプロジェクトで、サンプルと同じような書き方をすることは考えられません。コードを書き始めた日から3日もしないうちに、その非柔軟性が目に見えるものになるでしょう。そして、コードを書けば書くほど、不安定性が大きくなっていくという恐ろしい状況が待っています。また、そもそも数10万行のコードを2つや3つのファイルに収めることなどあり得ないので、考えられないというより、不可能と言えます。

それなのに、

「なぜ、普段通りに書かないのか？」

その理由は、

「可読性の向上ただ一つに尽きます」。

普段通り自由に書いていいのなら、むしろそのほうが早く書けますし、なにより楽です。

類書では、「自由に書いたもの」や、読みやすいとは到底思えない「ほとんどSDKサンプルそのまま」なものもあります。ファイルが多くあるので、切り替えるのも面倒だったり、過度のカプセル化をしたSDKをそのまま利用しているのを見ると、本当に読みやすさを考えているのだろうかと疑いたくなるものもあります。

学習用として、特に初心者には、このようなスタイルが優しいということが経験上分かっています。さらに、初心者でなくとも、数100行程度のサンプルコードであれば、逆にこのような書き方のほうが見やすい場合が多いと考えられるのも1つの理由になっています。

ちょっとしたプロジェクトであれば、サンプルのような書き方をするメリットは何もないのですが、500行以内程度のものであれば、積極的にこのスタイルをとるべきだと考えます。たとえ精神衛生上、決して良くはないグローバル変数を多用したとしてもです。

そのため、極力シンプルなコードを書くようにしています。ただし、シンプルにすることにより分かりづらくなったり、本質が薄れてしまうような場合は、シンプルさを犠牲にする場合もあります。本質を失わないで、かつ、分かりやすくシンプルに書くことを常に意識して書いています。一言で言えば「可読性」です。格好良いことを言っているようですが、そう意識しているのは確かであり、それによって普段の何倍も時間がかかるコーディングをしているということも事実です。実際そのように上手いこと書けているのかは読者の判断ではありますが、少なくとも自負はあります。

第1部
Direct3D
の
極意

DirectX
Visual C++ 2002/2003/2005対応
▶逆引き大全◀
500の
極意

第 1 章

001～225

レンダリングの極意

1-1　レンダリング準備と設定（007～020）
1-2　レンダリングの基本的な流れ（021～026）
1-3　スプライト（027～055）
1-4　3Dジオメトリ（056～121）
1-5　ジオメトリの質感作成・調整（122～182）
1-6　文字のレンダリング（183～184）
1-7　シーンの設定（185～204）
1-8　シーンの質感作成（205～225）

007 レンダリングに最低限必要なコードを書く

ウィンドウを作成して、Direct3Dを初期化するコードを書く　初級

固有サンプルプロジェクト　007_Direct3D最小コード

このサンプルは、ウィンドウを作成して、Direct3D関連の最低限の初期化までを行うものです。実行すると、黒い背景のみのシーンをレンダリングするウィンドウが現われます。

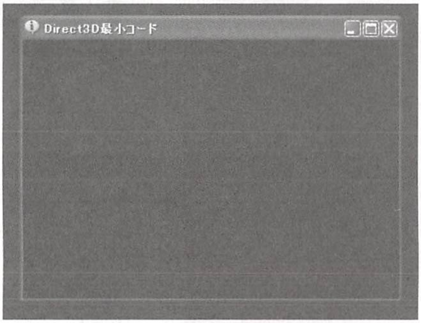
とにかく、何らかのレンダリングを行うのに最低限度必要な処理、言い換えると最低限度書かなくてはならないコードは次のとおりです。

1. ウィンドウを作成する
2. Direct3DオブジェクトとDirect3Dデバイスオブジェクトを作成する
3. 画面をクリアしてシーンを開始する

必須の処理はこの3つですが、4つ目として作成したDirect3Dオブジェクトのリリース（メモリからの開放）があります。

サンプルプロジェクトのソースコードMain.cpp内で該当する部分は次のとおりです。

▼基本サンプル実行画面



1．ウィンドウを作成する部分

```
22行　static char szAppName[] = "Direct3D最小コード";
(中略)
42行　UpdateWindow (hWnd);
```

このサンプルでは通常のウィンドウを作成していますが、フルスクリーンウィンドウ、あるいはダイアログウィンドウ（ダイアログボックスもウィンドウです）にもレンダリングできます。したがって、この部分は必ずしもこうである必要はありません。

2．Direct3DオブジェクトとDirect3Dデバイスオブジェクトを作成する

```
HRESULT InitD3d(HWND hWnd)
{
    // 「Direct3D」オブジェクトの作成
    if(NULL == (pD3d = Direct3DCreate9(D3D_SDK_VERSION)))
    {
          MessageBox(0,"Direct3Dの作成に失敗しました","",MB_OK);
    return E_FAIL;
    }
    // 「DIRECT3Dデバイス」オブジェクトの作成
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.BackBufferFormat =D3DFMT_UNKNOWN;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = TRUE;

    if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
          D3DCREATE_HARDWARE_VERTEXPROCESSING,
          &d3dpp, &pDevice)))
    {
          if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
hWnd,
                D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                      &d3dpp, &pDevice)))
          {
                MessageBox(0,"HALモードでDIRECT3Dデバイスを作成できません¥nREFモー
ドで再試行します",NULL,MB_OK);
                if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT,
D3DDEVTYPE_REF, hWnd,
                      D3DCREATE_HARDWARE_VERTEXPROCESSING,
                      &d3dpp, &pDevice)))
                {
                      if(FAILED(pD3d->CreateDevice
                            (D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, hWnd,
                            D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                                    &d3dpp, &pDevice)))
                      {
                            MessageBox(0,"DIRECT3Dデバイスの作成に失敗しました
",NULL,MB_OK);
                            return E_FAIL;
                      }
                }
          }
    }
    return S_OK;
  }
```

デバイスオブジェクト作成部分で、if文を4重にネストしているのでコードが長くなっていますが、それによりデバイスオブジェクトの作成に失敗する確率がほとんどなくなります。

3. 画面をクリアしてシーンを開始する部分

```
VOID Render()
{
    pDevice->Clear(0, NULL, D3DCLEAR_TARGET,D3DCOLOR_XRGB(30,30,30), 1.0f,
0);
    if(SUCCEEDED(pDevice->BeginScene()))
    {
           //
           //        この中で何らかのレンダリング関連のコードを書く
           //
           pDevice->EndScene();
    }
    pDevice->Present(NULL, NULL, NULL, NULL);
}
```

Clear関数で画面をクリアした後に、BeginScene関数とEndScene関数の間にレンダリングコードを書きます。BeginScene関数とEndScene関数で囲まれた部分以外にレンダリングコードを書いても失敗します。

One Point

レンダリングコードとは、スプライト、3Dプリミティブ、メッシュ等の何らかのジオメトリをレンダリングする処理です。レンダリングコードの書き方は様々なので、それぞれのTipsを参照してください。

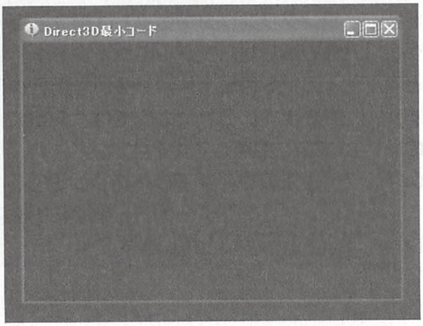
008 ウィンドウにレンダリングする

通常のウィンドウを作成し、その上にレンダリングする

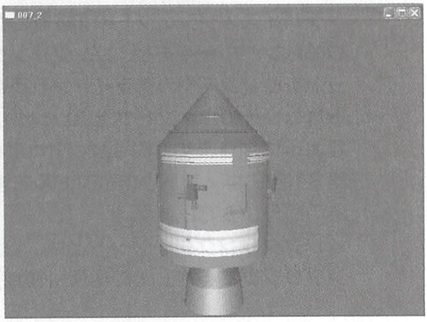
初級

固有サンプルプロジェクト　007_Direct3D最小コード
応用サンプルプロジェクト　007_2_ウィンドウにメッシュ

▼基本サンプル実行画面



▼応用サンプル実行画面



基本サンプルでは、まず通常のウィンドウを作成します。ウィンドウのメッセージプロシージャーは最低限の処理を書いています。(アプリケーションの終了ハンドラのみ)

ウィンドウ上にレンダリングするようになる直接のコードは次の部分です。

```
if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
    D3DCREATE_HARDWARE_VERTEXPROCESSING,
    &d3dpp, &pDevice)))
    {
    if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
    D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    &d3dpp, &pDevice)))
    {
          MessageBox(0,"HALモードでDIRECT3Dデバイスを作成できません¥nREFモードで再試
行します",NULL,MB_OK);
          if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF,
hWnd,
    D3DCREATE_HARDWARE_VERTEXPROCESSING,
    &d3dpp, &pDevice)))
          {
                if(FAILED(pD3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF,
hWnd,
    D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    &d3dpp, &pDevice)))
                {
                        MessageBox(0,"DIRECT3Dデバイスの作成に失敗しました",NULL,MB_OK);
                        return E_FAIL;
                }
                }
        }
```

デバイスオブジェクトの作成時にウィンドウのハンドルを渡すと、レンダリングターゲットをウィンドウのクライアント領域全体とみなし、そのウィンドウ上にレンダリングするようになります。CreateDevice関数の第3引数にウィンドウのハンドルを渡しているのが分かるでしょう。

レンダリングターゲットとは、「どこにレンダリングするか」その場所（領域）です。レンダリングターゲットはデフォルトでは引数で渡したハンドルのウィンドウ全体（クライアント領域全体）ですが、変更することもできます。それについては、Tips010から013を参照してください。

通常のウィンドウへのレンダリングでは、レンダリングターゲットを意識する必要がないことが多いと思います。

One Point

通常のウィンドウへのレンダリングでは、レンダリングターゲットを意識する必要がないことが多いと思います。

009　フルスクリーンウィンドウにレンダリングする

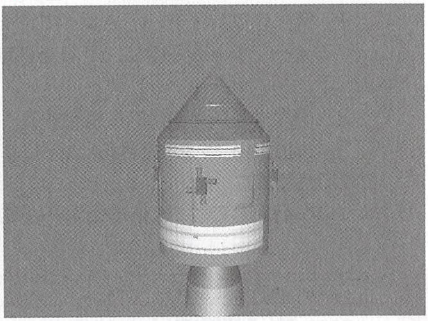
デバイスをフルスクリーンとして作成するとフルスクリーンになる　初級

固有サンプルプロジェクト　009_フルスクリーン
応用サンプルプロジェクト　009_2_フルスクリーンにメッシュ

フルスクリーンであろうとデスクトップ上のウィンドウであろうと、デバイスオブジェクトを作成する際には、事前にプレゼント・パラメーターを設定します。プレゼント・パラメーターの設定とはD3DPRESENT_PARAMETERS構造体のインスタンスを作り、その各メンバを初期化することです。

ウィンドウモードでは次のように初期化したりします。

▼応用サンプル実行画面。フルスクリーンのキャプチャ



```
// 「DIRECT3Dデバイス」オブジェクトの作成
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.BackBufferFormat =D3DFMT_UNKNOWN;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = TRUE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
```

ところがフルスクリーンの場合は、BackBufferFormatメンバにD3DFMT_UNKNOWを指定できません。フォーマット及び画面サイズその他を明示的に与えてやる必要があります。

この部分がウィンドウモードとフルスクリーンの（初期化における）相違点です。

```
// 「DIRECT3Dデバイス」オブジェクトの作成
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    D3DDISPLAYMODE DMode;
    pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT ,&DMode);
    d3dpp.BackBufferFormat =DMode.Format;
    d3dpp.BackBufferWidth=WINDOW_WIDTH;
    d3dpp.BackBufferHeight=WINDOW_HEIGHT;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = FALSE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat=D3DFMT_D16;
    d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT;
    d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE;
```

ウィンドウモードの設定と異なる部分は、

1．ディスプレイモードを調べて、そこからフォーマットを取得し、それを指定すること
2．画面サイズを明示的に指定すること
3．WindowedメンバはFALSEにすること

の3点です。

4つ目としてリフレッシュレートの設定をするのは任意です。

なお、フルスクリーンの場合は、「ウィンドウを閉じるボタン」が見えなくなるので、アプリケーションを終了させるハンドラが必要になります。サンプルでは次のように、「何らかのキーが押された場合」に終了するようにしています。

```
case WM_DESTROY:
case WM_KEYDOWN:
    PostQuitMessage(0)；
    break;
```

One Point

ここでのサンプルは、Alt+Tab等によりフルスクリーンからデスクトップ等に戻り、またフルスクリーンに復帰する時の復帰処理を実装していないので、フルスクリーンに復帰できません。復帰のコーディングはTips020を参照してください。

010 ダイアログ全体にレンダリングする

●モーダルダイアログのクライアント領域全体をレンダリングターゲットとする　初級

固有サンプルプロジェクト　010_ダイアログ全体
応用サンプルプロジェクト　010_2_ダイアログにメッシュ

ダイアログボックスはモーダルとモードレスがありますが、本Tipsはモーダルです。モードレスはTips012、013で解説しています。

モーダルダイアログの場合は、一旦ダイアログを作成したら、ダイアログ外部でメッセージをポーリングできず、全ての処理をダイアログのメッセージプロシージャーで処理しなければならないのが不便なところです。

そもそもモーダルダイアログは、一時的に何らかのメッセージを表示する程度に使用するので、通常は不便はありませんが、ここでのように内部にレンダリングするとなると、モーダルでは少々トリッキーなコードとなってしまいます。

ダイアログボックスのモードとレンダリング領域の組み合わせによる、特徴と留意点を一覧表にしたものが次の表です。

▼ダイアログへのレンダリング。パターン別の特徴と留意点

	モーダル	モードレス
全クライアント領域にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログで行う。他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。
特定の領域またはアイテム上にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログ内で行う。InvalidateRect関数は不要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える。InvalidateRect関数は不要。

モーダルなのでダイアログの作成はDialogBox関数で行います。SetTimer関数によりレンダリングを更新するタイミングを指定します。ここでは、60fps(毎秒60フレーム)となるようにしました。

```
//ダイアログの作成
     DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG1),NULL,(DLGPROC)DialogProc);
```

ダイアログを作成した瞬間から、ダイアログ外部では何もできなくなるので、WM_INITDIALOGハンドラでDirect3Dの初期化を行います。

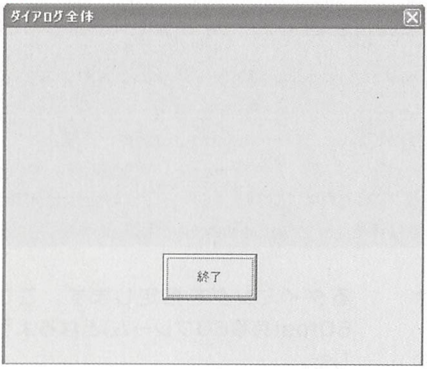
```
case WM_INITDIALOG:
            // ダイレクト3Dの初期化関数を呼ぶ
            if(FAILED(InitD3d(hDlg)))
            {
                    //アプリケーションを終了する時には、Direct3Dオブジェクトをリリースする
                    pDevice->Release();
                    pD3d->Release();
                    EndDialog(hDlg,TRUE);
            }
            SetTimer(hDlg,1,60/1000,NULL);
            return FALSE;
```

1/60秒（16.6ミリ秒）ごとにレンダリングを行います。

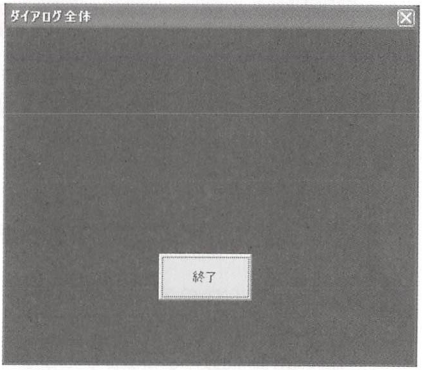
クライアント領域全体がレンダリングターゲットなので、そのままでは、ダイアログ内のアイテム（ボタン、リスト、スライダー等々）を塗り潰してしまいます。それを防ぐためにInvalidateRect関数により、ダイアログアイテムを再描画させます。

```
case WM_TIMER:
            Render();
            InvalidateRect(hDlg,NULL,NULL);
            return FALSE;
```

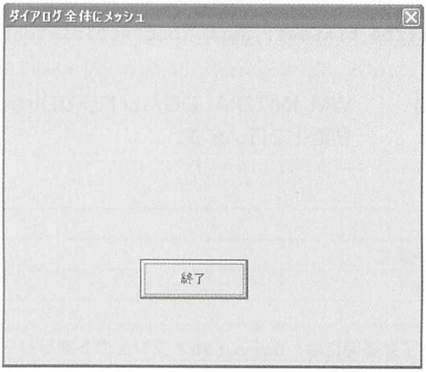
▼基本サンプル。レンダリングなし



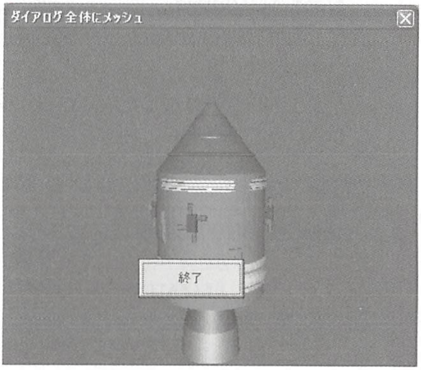
▼基本サンプル。レンダリングした状態



▼応用サンプル。レンダリングなし


ダイアログ全体にメッシュ
終了

▼応用サンプル。レンダリングした状態


ダイアログ全体にメッシュ
終了

One Point

モーダルダイアログは、ごく小規模な処理に使用しましょう。またクライアント領域全体にレンダリングすると他のアイテムがちらつく場合があります。

011 ダイアログの一部にレンダリングする

モーダルダイアログのクライアント領域内の指定領域をレンダリングターゲットとする

初級

固有サンプルプロジェクト　011_ダイアログの一部
応用サンプルプロジェクト　011_2_ダイアログの一部にメッシュ

ダイアログボックスはモーダルとモードレスがありますが、本Tipsはモーダルです。モードレスはTips012、013で解説しています。

モーダルダイアログの場合は、一旦ダイアログを作成したら、ダイアログ外部でメッセージをポーリングできず、全ての処理をダイアログのメッセージプロシージャーで処理しなければならないというのが不便なところです。

そもそもモーダルダイアログは、一時的に何らかのメッセージを表示する程度に使用するので、通常は不便はありませんが、ここでのように内部にレンダリングするとなると、モーダルでは少々トリッキーなコードとなってしまいます。

ダイアログボックスのモードとレンダリング領域の組み合わせによる、特徴と留意点を一覧表にしたものが次の表です。

▼ダイアログへのレンダリング。パターン別の特徴と留意点

	モーダル	モードレス
全クライアント領域にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログで行う。他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。
特定の領域またはアイテム上にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログ内で行う。InvalidateRect関数は不要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える。InvalidateRect関数は不要。

モーダルなのでダイアログの作成はDialogBox関数で行います。
SetTimer関数によりレンダリングを更新するタイミングを指定します。ここでは、60fps(毎秒60フレーム)となるようにしました。

```
//ダイアログの作成
    DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG1),NULL,(DLGPROC)DialogProc);
```

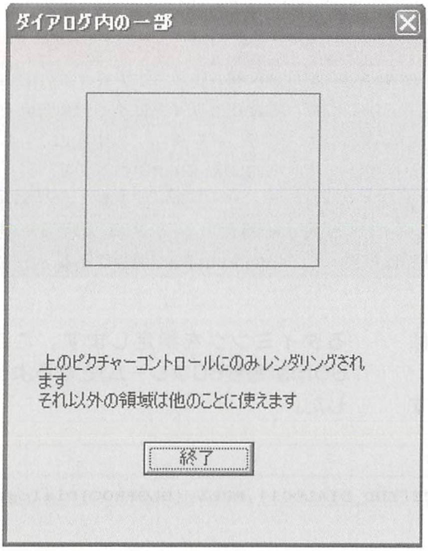
ダイアログを作成した瞬間から、ダイアログ外部では何もできなくなるので、WM_INITDIALOGハンドラでDirect3Dの初期化を行います。

```
case WM_INITDIALOG:
        // ダイレクト3Dの初期化関数を呼ぶ
        if(FAILED(InitD3d(hDlg)))
        {
            //アプリケーションを終了する時には、Direct3Dオブジェクトをリリースする
            pDevice->Release();
            pD3d->Release();
            EndDialog(hDlg,TRUE);
        }
        SetTimer(hDlg,1,60/1000,NULL);
        return FALSE;
```

1/60秒（16.6ミリ秒）ごとにレンダリングを行います。レンダリングはその領域のみのため、他のアイテムを塗り潰してしまうことはありませんので、InvalidateRect関数は不要です。

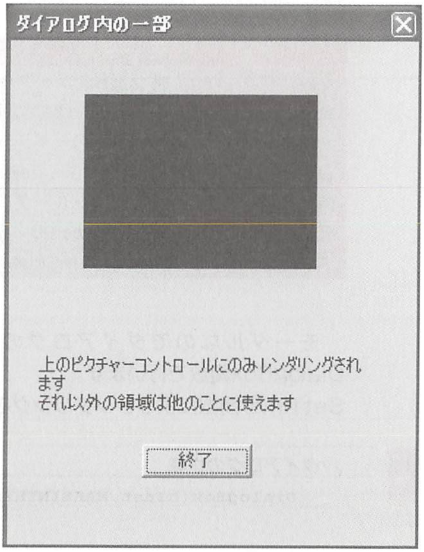
```
case WM_TIMER:
        Render();
        return FALSE;
```

▼基本サンプル。レンダリングなし



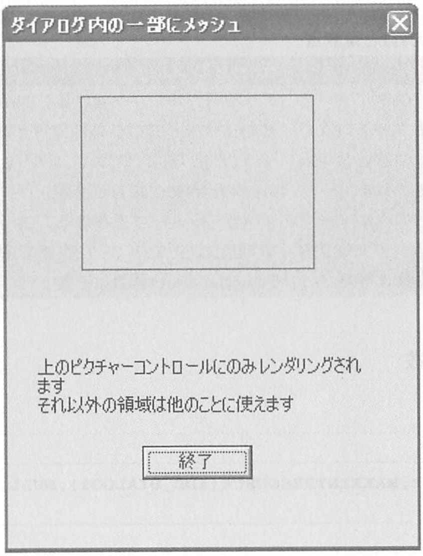
上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

▼基本サンプル。レンダリングした状態



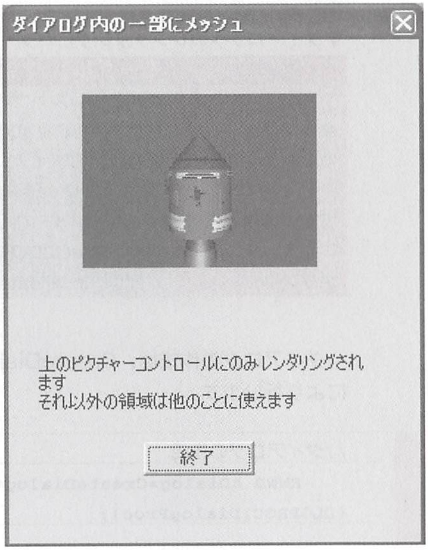
上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

▼応用サンプル。レンダリングなし



上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

▼応用サンプル。レンダリングした状態



上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

One Point

モーダルダイアログは、ごく小規模な処理に使用しましょう。ダイアログにレンダリングする場合、通常は領域を指定します。

012 モードレスダイアログ全体にレンダリングする

モードレスダイアログのクライアント領域全体をレンダリングターゲットとすることにより、ダイアログ全体にレンダリングする

初級

固有サンプルプロジェクト　012_モードレスダイアログ全体
応用サンプルプロジェクト　012_2_モードレスダイアログ全体にメッシュ

ダイアログボックスはモーダルとモードレスがありますが、本Tipsはモードレスです。モーダルはTips010、011で解説しています。

モードレスの場合はモーダルダイアログとは違い、ダイアログを作成しても、その後の行が実行されるのでダイアログ外部でメッセージをポーリングできます。

ダイアログボックスのモードとレンダリング領域の組み合わせによる、特徴と留意点を

一覧表にしたものが次の表です。

▼ダイアログへのレンダリング。パターン別の特徴と留意点

	モーダル	モードレス
全クライアント領域にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログで行う。他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。
特定の領域またはアイテム上にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログ内で行う。InvalidateRect関数は不要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える。InvalidateRect関数は不要。

ダイアログの作成は、CreateDialog関数により行います。

```
//ダイアログの作成
    HWND hDialog=CreateDialog(hInst,MAKEINTRESOURCE(IDD_DIALOG1),NULL,
(DLGPROC)DialogProc);
```

ダイアログを作成したら、次にDirect3Dの初期化を行います。

```
//ダイレクト3Dの初期化関数を呼ぶ
    if(FAILED(InitD3d(hDialog)))
    {
            return 0;
    }
```

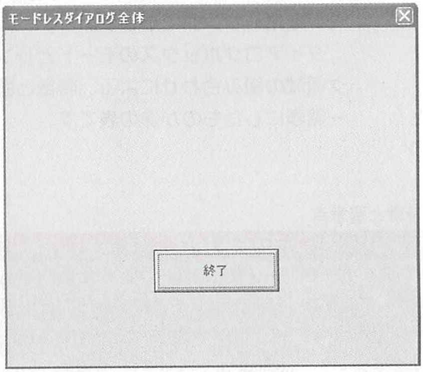
いわゆる通常のウィンドウの場合と同様にメッセージループを書きます。

クライアント領域全体がレンダリングターゲットなので、そのままでは、ダイアログ内のアイテム（ボタン、リスト、スライダー等々）を塗り潰してしまいます。それを防ぐためにRender関数の後にInvalidateRect関数により、ダイアログアイテムを再描画させます。

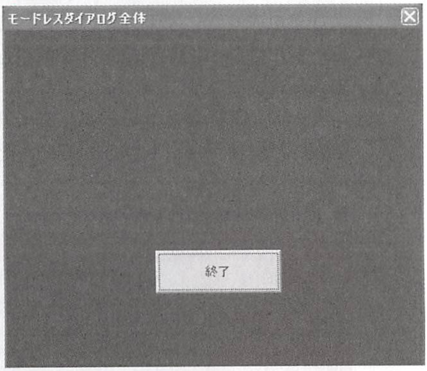
```
//メッセージループ
    MSG msg;
    ZeroMemory(&msg, sizeof(msg));
    while(msg.message!=WM_QUIT)
     {
            if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
            {
                    if(!IsDialogMessage(hDialog,&msg))
                    {
                            TranslateMessage(&msg);
                            DispatchMessage(&msg);
                    }
```

```
        }
        else
        {
            if(!g_boClose)
            {
                Render();
                InvalidateRect(hDialog,NULL,NULL);
            }
        }
    }
```

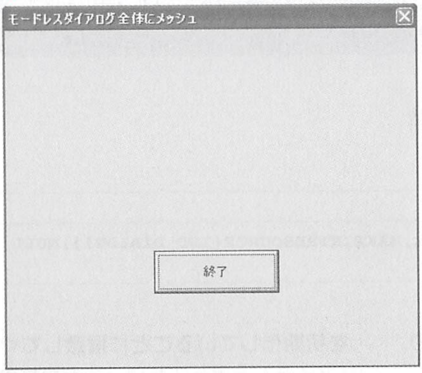
▼基本サンプル。レンダリングなし


モードレスダイアログ全体
終了

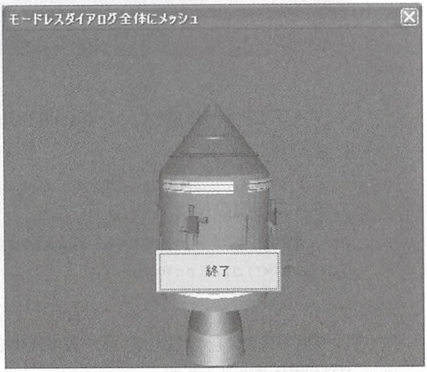
▼基本サンプル。レンダリングした状態


モードレスダイアログ全体
終了

▼応用サンプル。レンダリングなし


モードレスダイアログ全体にメッシュ
終了

▼応用サンプル。レンダリングした状態


モードレスダイアログ全体にメッシュ
終了

OnePoint

クライアント領域全体にレンダリングすると、他のアイテムがちらつく場合があります。

013 モードレスダイアログの一部にレンダリングする

●モードレスダイアログのクライアント領域内の指定領域をレンダリングターゲットとする

初級

固有サンプルプロジェクト　013_モードレスダイアログの一部
応用サンプルプロジェクト　013_2_モードレスダイアログの一部にメッシュ

ダイアログボックスはモーダルとモードレスがありますが、本Tipsはモードレスです。モーダルはTips010、011で解説しています。

モードレスの場合はモーダルダイアログとは違い、ダイアログを作成しても、その後の行が実行されすのでダイアログ外部でメッセージをポーリングできます。

ダイアログボックスのモードとレンダリング領域の組み合わせによる、特徴と留意点を一覧表にしたものが次の表です。

▼ダイアログへのレンダリング。パターン別の特徴と留意点

	モーダル	モードレス
全クライアント領域にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログで行う。他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える他のアイテムを塗りつぶさないように、InvalidateRect関数が必要。
特定の領域またはアイテム上にレンダリング	メッセージループの代わりにタイマーを利用。Direct3Dの初期化もダイアログ内で行う。InvalidateRect関数は不要。	メッセージループを書ける。Direct3Dの初期化はダイアログ外部で行える。InvalidateRect関数は不要。

ダイアログの作成は、CreateDialog関数により行います。

```
//ダイアログの作成
    HWND hDialog=CreateDialog(hInst,MAKEINTRESOURCE(IDD_DIALOG1),NULL,
(DLGPROC)DialogProc);
```

ダイアログを作成したら、次にDirect3Dの初期化を行います。

レンダリングターゲットにするアイテムのハンドルを得て、そのハンドルでDirect3Dを初期化していることに留意してください（アイテムと言ってもウィンドウなので、ウィンドウハンドルです）。

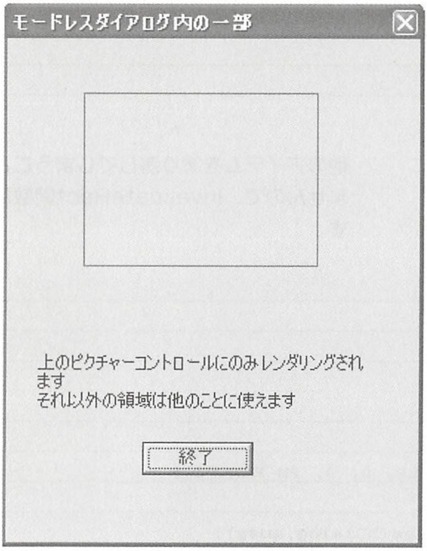
```
HWND hRenderTaret=GetDlgItem(hDialog,IDC_RENDER);
        // ダイレクト3Dの初期化関数を呼ぶ
        if(FAILED(InitD3d(hRenderTaret)))
        {
                return 0;
        }
```

いわゆる通常のウィンドウの場合と同様にメッセージループを書きます。

レンダリングはアイテム領域のみのため、他のアイテムを塗り潰してしまうことはありませんので、InvalidateRect関数は不要です。

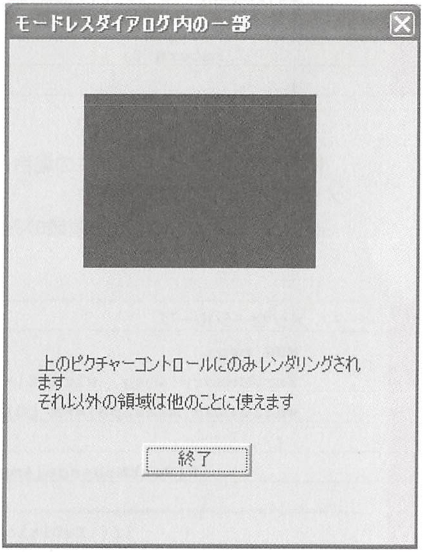
```
    // メッセージループ
        MSG msg;
        ZeroMemory(&msg, sizeof(msg));
        while(msg.message!=WM_QUIT)
         {
                if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
                {
                        if(!IsDialogMessage(hDialog,&msg))
                        {
                                TranslateMessage(&msg);
                                DispatchMessage(&msg);
                        }
                }
                else
                {
                        if(!g_boClose)
                        {
                                Render();
                                InvalidateRect(hDialog,NULL,NULL);
                        }
                }
         }
```

▼基本サンプル。レンダリングなし


モードレスダイアログ内の一部
終了

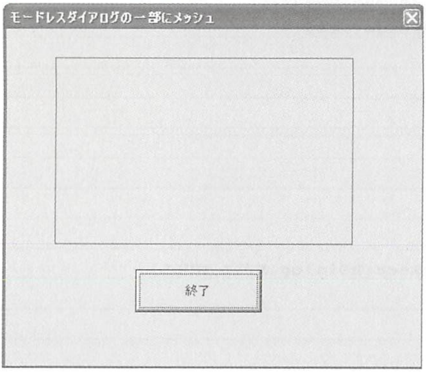
上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

▼基本サンプル。レンダリングした状態

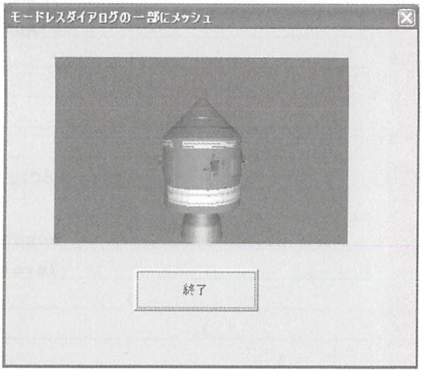

モードレスダイアログ内の一部
終了

上のピクチャーコントロールにのみレンダリングされます
それ以外の領域は他のことに使えます

▼応用サンプル。レンダリングなし


モードレスダイアログの一部にメッシュ
終了

▼応用サンプル。レンダリングした状態


モードレスダイアログの一部にメッシュ
終了

One Point

この組み合わせが、ダイアログ上のレンダリングとしてもっとも使えるものではないでしょうか。

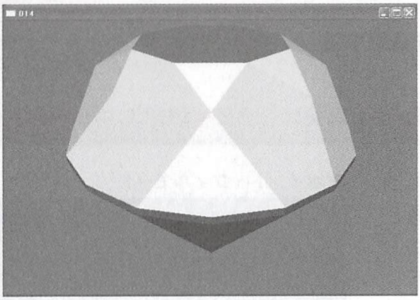
014 シェーディングモード(フラット、グーロー)の設定をする

●レンダリングステートのシェーディングモードを変更する

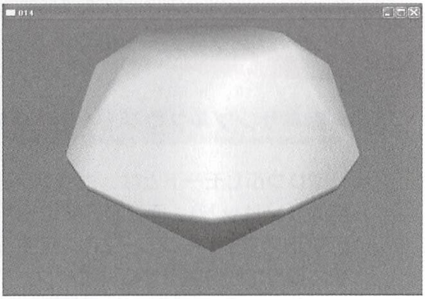
初級

固有サンプルプロジェクト 014_シェーディングモデル

▼フラットシェード



▼グーローシェード



フラットシェーディングは、各ポリゴンを単純に塗りつぶすものです。

グーローシェーディングは各ポリゴンの頂点法線を基にして、隣り合うポリゴン間に滑らかなシェード(陰影)を付けるという、フラットシェーディングよりも若干高級なモードです。

このサンプルのダイアモンドメッシュであれば、たまたまフラットシェーディングのほうが、面がくっきり見えて好都合なのですが、人の顔などの滑らかな表面を表現する時には、各ポリゴンがクッキリ見えては都合が悪く、その場合はグーローシェーディングにしたほうがいいでしょう。

サンプルの178から185行目で、IDirect3DDevice9::SetRenderStateメソッドによりシェーディングモードを指定しています。第1引数に、D3DRS_SHADEMODEを、第2引数に、フラットシェードの場合はD3DSHADE_FLAT を、グーローの場合はD3DSHADE_GOURAUDを渡します。グーローにする場合は、コメントアウトを解除してください。

pDeviceはIDirect3DDevice9のポインターです。

```
//シェーディングモデルを設定
	{
		//フラット・シェーディング
		pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_FLAT);

		//グーロー・シェーディング
		//pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_GOURAUD);
	}
```

One Point

Direct3Dのデフォルトはグーローなので、何も指定しない場合はグーローシェーディングされます。フラットは単純なモデルのため高速なシェーディングを行うのが特徴です。

015 塗りつぶしモード（ワイヤーフレーム、ソリッド等）の設定をする

レンダリングステートのフィルモードを変更する

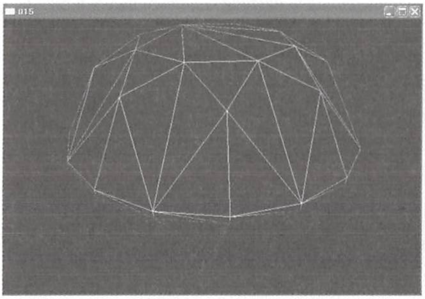
初級

固有サンプルプロジェクト　015_フィルモード

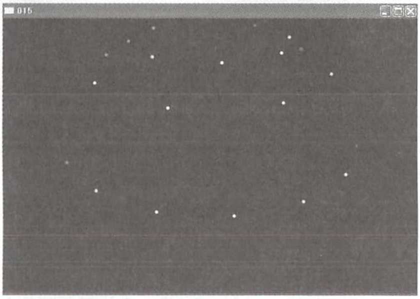
塗りつぶしモードとは、ポリゴン面に色を塗る（ソリッド）、面を塗らずに辺だけを表示する（ワイヤーフレーム）、辺すら表示せずに頂点だけを表示する（ポイント）モードのことで、フィルモードとも言います。

サンプルの179から184行目で、IDirect3DDevice9::SetRenderStateメソッドによりフィルモードを指定しています。第1引数に、D3DRS_FILLMODEを、第2引数に、ポイントの場合はD3DFILL_POINTを、ワイヤーフレームの場合はD3DFILL_WIREFRAMEを、ソリッドの場合はD3DFILL_SOLIDを指定します。

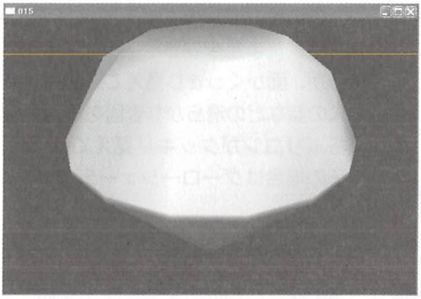
▼ワイヤーフレーム・フィルモード



▼ポイント・フィルモード



▼ソリッド・フィルモード



```
//フィルモード（塗りつぶしモード）の設定
	{
        //pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_POINT);
        pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_WIREFRAME);
        //pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_SOLID);
    }
```

pDeviceはIDirect3DDevice9のポインターです。

変えたいモードの行のコメントアウトを解除してください（他2つの行はコメントアウトしてください）。

One Point

Direct3Dはデフォルトでソリッドモードです。

016 カリングの設定をする

●ポリゴンの裏側をレンダリングしないことにより高速化を図る

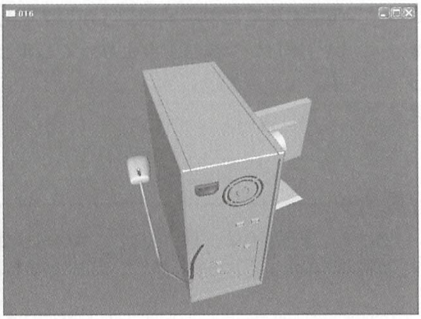
初級

固有サンプルプロジェクト　016_カリングによる違い

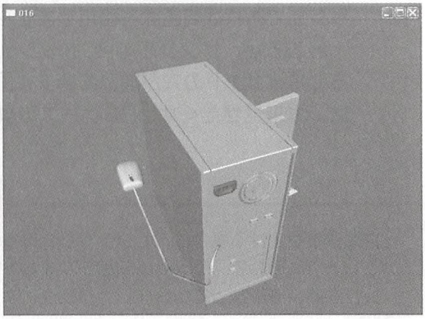
カリングとは、ポリゴンの裏面をレンダリングしないことにより、処理速度を向上させることを言います。カリングの由来は、「カル（CULL）」することであり、CULLとは「選り抜く、抜粋する」という意味です。ポリゴンの表のみを選り抜き、レンダリングするというわけです。

サンプルの１７８から１８２行目で、IDirect3DDevice9::SetRenderStateメソッドによりカリングするかしないかを指定しています。下のコードはカリングしないように設定しています。

▼カリングしない（両面レンダリング）



▼カリングする（表面のみレンダリング）



```
//カリング（陰面非描画）の設定
    {
            pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
            //pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
    }
```

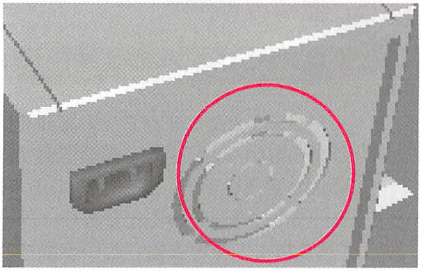
カリングしないということは、言い換えれば、裏面もレンダリングするということです。レンダリング結果に、何もおかしな部分はありません。

次に、カリングするようにコメントアウトします。

```
//カリング（陰面非描画）の設定
    {
            //pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
            pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
    }
```

カリングした時の実行結果を見てください。わかりづらいかもしれませんが、妙な部分が1箇所あります。妙な部分を拡大したのが右の図です。

▼カリングした場合の不具合ケースの裏面がレンダリングされないので、見えないはずの液晶モニターが透けて見える



裏面をレンダリングしなくても、ポリゴンが全てこちらを向いている、つまり閉じたメッシュであれば問題は表面化しません。

ところが、サンプルのPCメッシュは、後ろに排気ダクトの穴が開いています。穴からは、本来見せたくないPCケースの裏面が見える格好になります。裏面はカリングによりレンダリングされないので、ケースの向こう側にあるモニターメッシュが見えてしまうという妙なレンダリングとなってしまっています。

One Point

描画速度を上げる場合は、カリングしたほうがいいでしょう。そして、Direct3Dはデフォルトでカリングするようになっています。カリングによって透けてしまう場合は、透けてほしくないポリゴンに反対向きのポリゴンを貼るなどして工夫しましょう。

017 アンビエント光（環境光）を設定する

簡単に全体的な照度を調整できるアンビエントライト

初級

固有サンプルプロジェクト　017_アンビエントライト

アンビエントライトは、環境光のことです。環境光は“ライト“というよりも”画面全体の明るさ（と色合い）“程度のものでCG特有のものです。現実のライトをシミュレートしているわけではなく、ただ単純に全体的な明るさや色合いを画一的に調整するだけのものです。　環境光は、凹凸のあるメッシュでも陰影は計算されないため、ベタ塗り状態になっていしまい、また、方向がないため、どのような場所にあっても同様にライティングされてしまいます。

アンビエントライトに対し、ダイレクトライト（ポイント、スポット、ディレクショナル）があります。ダイレクトライトは、よりリアルなライティングを行うものでTips018で解説しています。

サンプルの180行目で、IDirect3DDevice9::SetRenderStateメソッドによりアンビエントライトの強度を指定しています。

```
pDevice->SetRenderState(D3DRS_AMBIENT, 0x55555555);
```

第1引数にD3DRS_AMBIENTを第2引数にライトの強度と言いましたが、具体的にはARGB値を指定します。ARGB値は、アルファ値+赤値+緑値++青値から成る32ビット値なので、16進数で例えば0xaaaaaaaaと書いてもいいですし、10進数で書いてもいいのですが、10進数の場合は、見た目に何色成分がどれくらいの割合なのかということが分からないので、通常は16進数で書きます。または、もっと簡単にARGB()マクロを次のように使用してもいいでしょう。

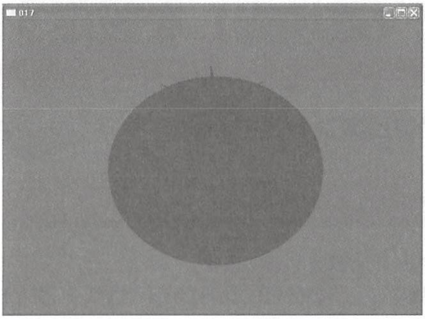
```
pDevice->SetRenderState(D3DRS_AMBIENT, D3DCOLOR_ARGB(255,0,255,0));
```

この場合は、アルファ成分=不透明、赤成分=0、緑成分=255（最大)、青成分=0なので、ライトの“色”が緑になります。つまり、第2引数は、ライトの明るさとライトの色の両方を意味しているのです。

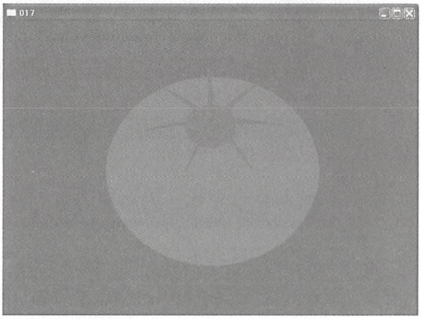
サンプルでいろいろ試してみてください。次の図は、強度0x55555555によるレンダリング結果です。強度を均一に増加させると（明るくすると)、次のように明るくレンダリングされます。ただし、各成分を均等に増加させているので、もとのトマトの“色合い“はそのままです。

また、アンビエント光は、もっとも単純な照明モデルであり、シェード（陰影）やスペキュラー（光沢）は計算されません。その結果、図のように“ベッタリ”したレンダリングになり、まるで2D絵のようです。いまひとつ見栄えが良くありません。

▼強度0x55555555



▼強度0xffffffff（強度を均一に増加）



One Point

アンビエントライトに頼るライティングはやめましょう。全体に“のっぺり”したレンダリングになってしまいます。極端な話、アンビエントをゼロにして、他の種類のライトだけで照明することもよく行われる手法です。

018 色々なライトを作る

●ライトの種類（ポイント、スポット、ディレクショナル）と属性（ディフューズ、スペキュラー）を指定して、さまざまなライトを作り出す

初級

固有サンプルプロジェクト　018_1_ディレクショナルライト
固有サンプルプロジェクト　018_2_ポイントライト
固有サンプルプロジェクト　018_3_スポットライト

アンビエントライト（厳密に言えばライトではありませんが…）に対して、本当の意味でのライトをDirect3Dでは“ダイレクトライト”と言います。

まずは種類ですが、ダイレクトライトには3種類あります。ディレクショナルライト、ポイントライト、スポットライトの3つです。それぞれの特徴は次の表のとおりです。

▼ダイレクトライトの特徴とパラメータの違い

	概要	明るさ	色	位置	方向	減衰	フォールオフ	コーン角度	計算速度
ディレクショナル	平行無限遠光源、太陽光	○	○	×	○	×	×	×	速い
ポイント	点光源	○	○	○	×	○	○	×	普通
スポット	そのまんま	○	○	○	○	○	○	○	遅い

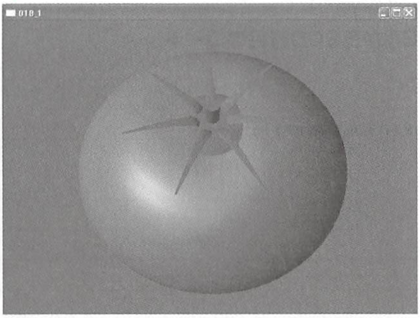
明るさと色は全てのタイプのライトが持っている属性です。ディレクショナルライトは方向のみを持ち、ポイントライトは方向は持たず、位置のみが意味を持ちます。スポットライトは、全ての属性を持ち、最もドラマチックな効果を出すことのできる高級なライトですが、そのため、計算には最も時間がかかります。

それぞれのライトタイプのレンダリング効果を見ていきましょう。

●018_1_ディレクショナルライト　サンプルの182から196行目のコード

```
D3DXVECTOR3 vecDirection(0,-1,1);
          D3DLIGHT9 light;
          ZeroMemory(&light, sizeof(D3DLIGHT9));

          light.Diffuse.r  =1.0f;
          light.Diffuse.g  =1.0f;
          light.Diffuse.b  = 1.0f;
          light.Specular.r=1.0f;
          light.Specular.g=1.0f;
          light.Specular.b=1.0f;
          //ディレクショナルライト固有の属性
          {
                light.Type       = D3DLIGHT_DIRECTIONAL;
                light.Direction=D3DXVECTOR3(1,-1,1);
          }
```

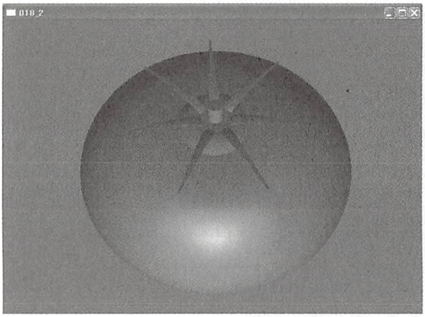
▼ディレクショナルライト（ディフューズ、スペキュラー共に普通）



●018_2_ポイントライト　サンプル　182から197行目のコード

```
D3DLIGHT9 light;
          ZeroMemory(&light, sizeof(D3DLIGHT9));

          light.Diffuse.r  =1.0f;
          light.Diffuse.g  =1.0f;
          light.Diffuse.b  = 1.0f;
          light.Specular.r=1.0f;
          light.Specular.g=1.0f;
          light.Specular.b=1.0f;
          //ポイントライト固有の属性
          {
                light.Type      = D3DLIGHT_POINT;
                light.Attenuation0 = 1.0f; //ポイントライトでattenuate（減衰：ア
テニュエート）させるとクラッシュする可能性がある。=1とすること
                light.Position=D3DXVECTOR3(0,0,-2);
                light.Range     = 20.0f;
          }
```

▼ポイントライト（ディフューズ、スペキュラー共に普通）

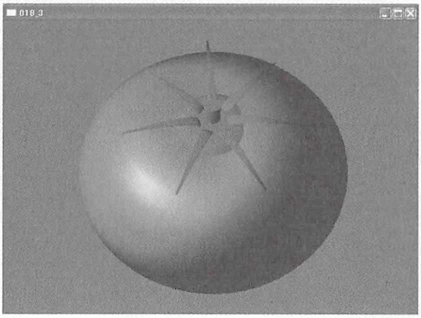


●018_3_スポットライト　サンプル　181から200行目のコード

```
D3DLIGHT9 light;
          ZeroMemory(&light, sizeof(D3DLIGHT9));

          light.Diffuse.r =1.0f;
          light.Diffuse.g  =1.0f;
          light.Diffuse.b  = 1.0f;
          light.Specular.r=1.0f;
          light.Specular.g=1.0f;
          light.Specular.b=1.0f;
          //スポットライト特有の属性
          {
                light.Type      = D3DLIGHT_SPOT;
                light.Attenuation0 = 1.0f; //スポットライトでattenuate（減衰：ア
テニュエート）させるとクラッシュする可能性がある。=1とすること
                light.Position=D3DXVECTOR3(-2,2,-2);
                light.Direction=D3DXVECTOR3(1,-1,1);
                light.Range     = 20.0f;
```

```
                    light.Theta=D3DX_PI/12;//θは内部コーン角度
                    light.Phi=D3DX_PI/10;//φは外部コーン角度 θとφの大小を間違えるとク
ラッシュする可能性あり
                    light.Falloff=1.0f;
            }
```

▼スポットライト（ディフューズ、スペキュラー共に普通）



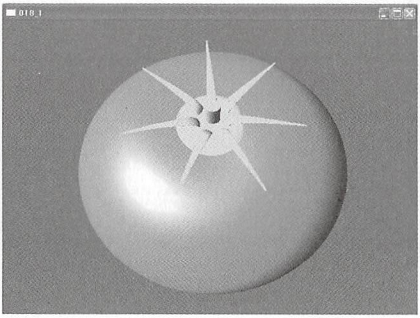
ダイレクトライトは、主にディフューズ（拡散反射）、スペキュラー（鏡面反射）というパラメーターを持ち、これらを変更することにより、シェード（陰影）やハイライト（てかり）を調整できます。

次3つの図は、ディフューズとスペキュラーを最大にした時のレンダリングです。先の図と比較すると、効果が分かると思います。

●ディフューズ大（タイプはディレクショナルライト）

```
light.Diffuse.r  =3.0f;
          light.Diffuse.g  =3.0f;
          light.Diffuse.b  = 3.0f;
          light.Specular.r=1.0f;
          light.Specular.g=1.0f;
          light.Specular.b=1.0f;
```

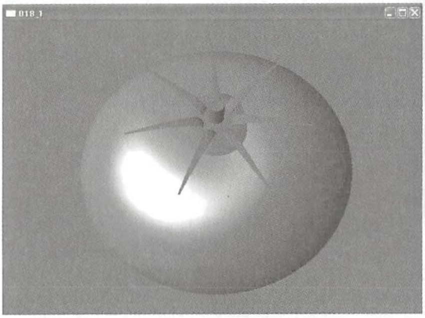
▼ディフューズ大（タイプはディレクショナルライト）



●スペキュラー大（タイプはディレクショナルライト）

```
light.Diffuse.r  =1.0f;
          light.Diffuse.g  =1.0f;
          light.Diffuse.b  = 1.0f;
          light.Specular.r=3.0f;
          light.Specular.g=3.0f;
          light.Specular.b=3.0f;
```

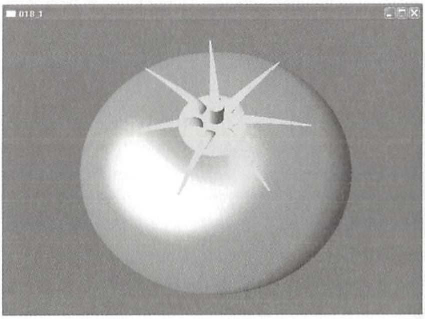
▼スペキュラー大（タイプはディレクショナルライト）



●ディフューズ、スペキュラー共に大（タイプはディレクショナルライト）

```
light.Diffuse.r  =3.0f;
          light.Diffuse.g  =3.0f;
          light.Diffuse.b  = 3.0f;
          light.Specular.r=3.0f;
          light.Specular.g=3.0f;
          light.Specular.b=3.0f;
```

▼ディフューズ、スペキュラー共に大（タイプはディレクショナルライト）



One Point

アンビエントライティングとは全く違う、リアルなライティングであることがお分かりでしょう。スポットライトは重たいライトなので、多数必要な時には、慎重に使用するようにしましょう。

019 アルファブレンド（透過処理）の設定をする

●フレームバッファアルファによりジオメトリを半透明にする

中級

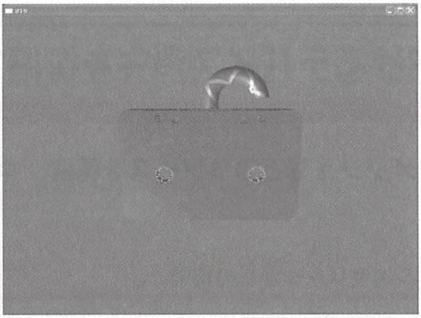
固有サンプルプロジェクト　019_フレームバッファアルファブレンド

アルファブレンドブレンドとは、あるピクセルを画面のある場所に描画する瞬間、そこの場所にもともと描画されていたピクセルと色を混ぜ合わせることにより、透明感を出す動作であり、その技法をアルファブレンディングと呼びます。

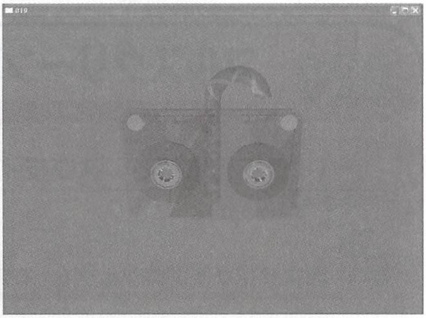
アルファブレンディングは、頂点単位、マテリアル単位、テクスチャ単位、レンダリングターゲット単位に行うことができます。

サンプルでは、もっとも直感的なフレームバッファアルファ、つまり画面単位のアルファブレンディングを行っています。

▼アルファブレンドなし



▼アルファブレンドあり



レンダリングステートの設定で、透明なメッシュだろうが不透明なメッシュだろうが透けるようになります。“透明なメッシュ”とは、透明なマテリアルであるメッシュのことです。透明なマテリアルとは、アルファ値が0より大きく1.0未満のマテリアルです。

```
//アルファブレンドの設定
{
        pDevice->SetRenderState (D3DRS_ALPHABLENDENABLE, TRUE);
        pDevice->SetRenderState (D3DRS_SRCBLEND, D3DBLEND_ SRCCOLOR);
```

```
            pDevice->SetRenderState (D3DRS_DESTBLEND, D3DBLEND_ INVSRCCOLOR);
            pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1,D3DTA_DIFFUSE);
            pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG2,D3DTA_DIFFUSE);
      }
```

マテリアルが透明でないメッシュも透けます。というのは、何を基準にブレンドするかという部分でカラーを指定しているからです。

```
pDevice->SetRenderState (D3DRS_SRCBLEND, D3DBLEND_SRCCOLOR);
pDevice->SetRenderState (D3DRS_DESTBLEND, D3DBLEND_INVSRCCOLOR);
```

マテリアルを反映させたい場合は、D3DBLEND_ SRCALPHA, D3DBLEND_ INVSRALPHAとします。

```
pDevice->SetRenderState (D3DRS_SRCBLEND, D3DBLEND_ SRCALPHA);
pDevice->SetRenderState (D3DRS_DESTBLEND, D3DBLEND_ INVSRALPHA);
```

One Point

マテリアルアルファの場合は、（メッシュの）マテリアルのアルファ値自体を動的に変えれば、透明感の調整ができます。いろいろ試してください。

020 フルスクリーンにおけるデバイス消失を復旧する

●デバイスを再取得し、スプライトやメッシュなどのリソースを再構築する 中級

固有サンプルプロジェクト 020_フルスクリーンからの復帰

フルスクリーンの場合は、デバイスを失う危険性が常にあります。一方、ウィンドウモードの場合、デバイスを失うことは基本的にありません。

「デバイスを失う」とはなんでしょう？

デバイスを失うと、レンダリング全てが失敗します。例えば、ゲーム中に何らかの理由により、ゲームの進行はそのままでデスクトップに戻りたい場合があります。その場合、Alt＋Tabによりデスクトップに切り替えて、用事が済んだらまたゲーム内に戻るでしょう。その時、復帰処理コードが書かれていないと、ゲーム画面は真っ暗になるか、目茶苦茶なピクセルが表示されるか、あるいはデスクトップ画面の断片が表示されるかの何れかになます。（グラフィックがおかしいだけで、ゲームは進行しているし、音も出ているかもしれません）

ウィンドウとフルスクリーンでなぜ違うのでしょう？

フルスクリーンアプリケーションが画面（フレームバッファ等）を占有してしまうことにより発生します。ウィンドウモードの場合は、デスクトップ上で他のアプリケーションやアイコン、要するにデスクトップに表示されるグラフィックとうまく協調するようにデスクトップの管理下にあります。フルスクリーンの場合は、デスクトップ上のグラフィックが格納されているメモリを食いつぶし、それまでデスクトップに表示されていたグラフィックを壊してしまいます。これは同時に、フルスクリーンからデスクトップに戻った時（Alt+Tabでデスクトップに切り替えた時）に、今度はデスクトップがフルスクリーンアプリのグラフィックを壊してしまうことを意味します。デスクトップとフルスクリーンは同じメモリ領域（バックバッファ）を使用するので、一方が表示されれば、もう片方のピクセルが壊れる（上書きされてしまう）ということです。

フルスクリーンアプリケーションあるいは、フルスクリーンになり得るアプリケーションは、デバイス消失を常に監視しなくてはなりません（市販ゲームでもお粗末なものは、この対処がなされていないものもありました）。

デバイス消失の判断は、次のように画面フリップ処理の成否で判断します。

```
if(pDevice->Present(NULL, NULL, NULL, NULL)==D3DERR_DEVICELOST)
```

そして、消失している場合の対処は次のように行います。

1. まず、「再構築できる状態」かどうかを判断しなくてはなりません。

```
if(pDevice->TestCooperativeLevel() == D3DERR_DEVICELOST)
```

2. 再構築できる状態であれば、再構築します。できない状態であれば、できる状態になるまで待ちます（再構築を行わないで、そのままリターンするなど）。再構築とは、デバイスをリセットして、次のようにすることです。

```
D3DPRESENT_PARAMETERS d3dpp;

        ZeroMemory(&d3dpp, sizeof(d3dpp));
        D3DDISPLAYMODE DMode;
        pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT ,&DMode);
        d3dpp.BackBufferFormat =DMode.Format;
        d3dpp.BackBufferWidth=WINDOW_WIDTH;
        d3dpp.BackBufferHeight=WINDOW_HEIGHT;
        d3dpp.BackBufferCount=1;
        d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
        d3dpp.Windowed = FALSE;

        if(pDevice->Reset(&d3dpp)!=D3D_OK)
```

さらに、グラフィックリソース（スプライト、テクスチャ、サーフェイス、メッシュ等）を再度作成することです。次のコードはスプライトとテクスチャの再作成ですが、メッシュでも同様です。再度LoadMeshFromFile関数等により、初期化時と同じコードを書くことになります。

```
//「テクスチャオブジェクト」の“再”作成
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite.bmp",SPRITE_WIDTH,SPRI
TE_HEIGHT,0,0,   D3DFMT_A8R8G8B8,

D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,

0xff000000,NULL,NULL,&pTexture)))
          {
                MessageBox(0,"テクスチャの再構築に失敗しました","",MB_OK);
                return ;
          }
          // 「スプライトオブジェクト」の”再”作成
          if(FAILED(D3DXCreateSprite(pDevice,&pSprite)))
          {
                MessageBox(0,"スプライトの再構築に失敗しました","",MB_OK);
                return ;
          }
```

以下は、サンプルプログラムの180から220行目です。デバイス消失への対処部分の全てになります。

```
//デバイス消失への対応
    if(pDevice->Present(NULL, NULL, NULL, NULL)==D3DERR_DEVICELOST)
    {
          if(pDevice->TestCooperativeLevel() == D3DERR_DEVICELOST)
          {
                return;
          }
          SAFE_RELEASE(pTexture);
          SAFE_RELEASE(pSprite);

          D3DPRESENT_PARAMETERS d3dpp;

          ZeroMemory(&d3dpp, sizeof(d3dpp));
          D3DDISPLAYMODE DMode;
          pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT ,&DMode);
          d3dpp.BackBufferFormat =DMode.Format;
          d3dpp.BackBufferWidth=WINDOW_WIDTH;
          d3dpp.BackBufferHeight=WINDOW_HEIGHT;
          d3dpp.BackBufferCount=1;
          d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
          d3dpp.Windowed = FALSE;

          if(pDevice->Reset(&d3dpp)!=D3D_OK)
          {
                MessageBox(0,"デバイスのリセットに失敗しました","",MB_OK);
          }
          //「テクスチャオブジェクト」の”再”作成
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite.bmp",SPRITE_WIDTH,SPRI
TE_HEIGHT,0,0,   D3DFMT_A8R8G8B8,

D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,

0xff000000,NULL,NULL,&pTexture)))
          {
                MessageBox(0,"テクスチャの再構築に失敗しました","",MB_OK);
                return ;
          }
          //  「スプライトオブジェクト」の”再”作成
          if(FAILED(D3DXCreateSprite(pDevice,&pSprite)))
          {
                MessageBox(0,"スプライトの再構築に失敗しました","",MB_OK);
                return ;
          }

    }
```

One Point

要するに、デバイスをリセットして、初期化コードをそのままコピーするか、初期化ルーチンを再度呼び出せばいいわけです。

021 画面をクリアする

●IDirect3Ddevice9::Clearメソッドを使う

初級

画面のクリアは厳密に言えば、画面を“単色で塗りつぶす”ことです。画面のクリアをしないと、前回のフレームに描画されたグラフィックが残ったままになり、移動している絵は残像のようになってしまいます。

クリアはIDirect3Ddevice9::Clearメソッドにより行い、引数は大抵次のようになります。塗りつぶす色は、この例では黒っぽい灰色としています（D3DCOLOR_XRGB(30,30,30)）。

```
IDirect3Ddevice9::Clear(0,NULL,D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,D3DCOLOR_XRG
B(30,30,30), 1.0f, 0);
```

クリア処理の直接的コードはこれだけですが、このクリアを成功させるには前提が必要です。それは、Zバッファーを有効にしていることと、

```
IDirect3Ddevice9::SetRenderState(D3DRS_ZENABLE, TRUE);
```

デバイス作成時に次のように設定していることです。

```
d3dpp.EnableAutoDepthStencil = TRUE;
d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
```

(d3dppはD3DPRESENT_PARAMETERS型のインスタンスとします)

それらをしていないと、このクリアメソッドは失敗します。クリアする色を変えてみてください。

022 シーンを開始・終了する

●レンダリング関係のメソッドは有効なシーン内でのみ成功する

初級

シーンとは何でしょう？
意味の広い言葉ですが、Direct3DにおけるAPI解説の文中では「何らかのレンダリングを行うコード部分及びそのタイミング」と

考えてください。

単にレンダリングコードを書いただけでは、有効なシーンとはなりません。たとえば、スプライトをレンダリングするためにDrawSprite()関数を実行したり、メッシュをレンダリングするためにDrawSubset()関数を書いただけでは、それらの関数は失敗します。

有効なシーンにするためには、IDirect3DDevice9::BeginScene メソッドによりシーンを明示的に開始し、そして、シーンの終わりにはIDirect3DDevice9::EndScene メソッドを実行しなくてはなりません。つまり、レンダリングコードはBeginSceneとEndSceneで囲まなければならないということです。

```
IDirect3DDevice9::BeginScene()

(この部分で何らかのレンダリングを行う)

IDirect3DDevice9::EndScene()
```

この範囲外で、レンダリングを行おうとしても失敗します。平たく表現すると、BeginSceneはDirect3Dに「今からレンダリングするから準備してね！」と指示していることであり、そしてEndSceneは「もうレンダリングの必要はないから、ゆっくり休んでよ」と伝えていることです。

なお、レンダリングが成功しても、画面を更新しなければ、画面に見えるようになりません。画面の更新とは、バックバッファとフロントバッファを切り替える（フリップ）ことです。それはIDirect3DDevice9::Presentメソッドにより行います。

One Point

BeginSceneとEndSceneで形成されるブロック（囲まれる部分）は、ネスト（入れ子）できません。BeginScene を連続して呼び出すと2回目の呼び出しは失敗します。

023 ワールド変換をする

●ジオメトリをワールド座標系に配置する　初級

固有サンプルプロジェクト　023_ワールド変換の効果

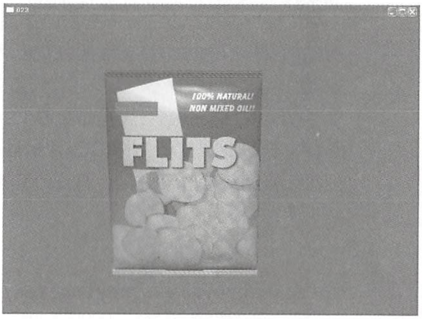
3Dジオメトリをディスプレイという2次元平面にレンダリングする時は、ワールド変換、ビュー変換、プロジェクション変換という3つの変換がこの順番で毎回必ず行われます。

ビュー変換については、Tips024、プロジェクション変換についてはTips025を参

照してください。

ワールド変換は一番最初に行われる変換であり、その目的は、3Dジオメトリを絶対座標系上に配置することです。配置するということは、位置はもちろん、回転（姿勢）、拡大縮小も含まれます。

▼ワールド変換コードを書かない場合



このサンプルでカメラの位置は－6単位後ろなので、それなりにレンダリングされますが、メッシュの位置は原点、回転なし、スケーリングなしというワールド変換がされてしまいます。

コード上、ワールド変換は次のようになります。

```
D3DXMATRIX matWorld;

(matWorldにワールド行列としての値を埋める)

IDirect3DDevice9::SetTransform(D3DTS_WORLD , &matWorld);
```

※注意　当然matWorldは適当に考えたインスタンス名なので、名前は任意です。

では、どうやってワールド行列の値を埋めるのか（行列を作成するのか）？

●位置

まずは、絶対座標系の意図した位置に配置するワールド行列の作成から覚えましょう。特定の位置に配置する行列は移動行列でもあります。なぜなら移動は連続した配置に他ならないからです。配置と移動は根本的に同じ動作です。

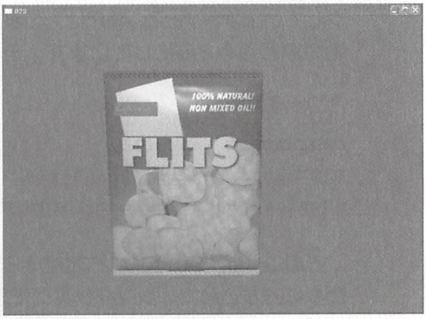
位置行列（移動行列）はD3DXMatrixTranslation関数により作成します。次のコード片は、絶対座標系の座標(3,1,7)にジオメトリを配置します。

▼コード片1

```
D3DXMATRIX matWorld;
D3DXMatrixTranslation(&matWorld,3,1,7);
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
```

このコードを実際にサンプルの195行目あたりに打ち込んで、レンダリングを確認してください。また、座標を変更してその効果を確認してください。

▼コード片1のワールド変換を設定した場合



●回転

次は回転です。ある瞬間の回転状態は姿勢とも呼ばれます。3次元空間での回転は3軸周りそれぞれに有り得るので、回転行列（または姿勢行列）を作成する関数も3種類あります。

D3DXMatrixRotationX関数（X軸回りの回転）
D3DXMatrixRotationY関数（Y軸回りの回転）
D3DXMatrixRotationZ関数（Z軸回りの回転）

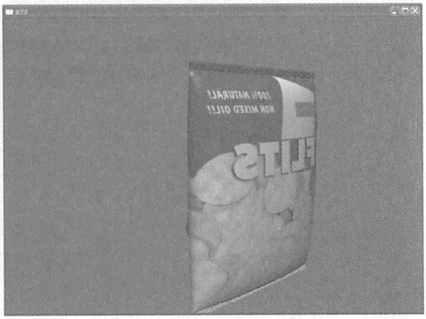
次のコード片は、ジオメトリをY軸回りに回転させます。回転の速度は3000分の1ラジアン/ミリ秒です。

▼コード片2

```
D3DXMATRIX matWorld;
D3DXMatrixRotationY(&matWorld, timeGetTime()/3000.0f);
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
```

このコードを実際にサンプルの195行目あたりに打ち込んで、レンダリングを確認してください。また、数値を変更して、その効果を確認してください。絵では分かりませんが、回転しています。

▼コード片2のワールド変換を設定した場合



●スケール（拡大縮小）

最後はスケールを変化させるスケーリング行列を作成しましょう。

スケーリング行列は、D3DXMatrixScaling関数に3つのFLOAT値を渡して作成します。3つのFLOAT値は、X軸に対する縮尺、Y軸に対する縮尺、Z軸に対する縮尺を意味します。

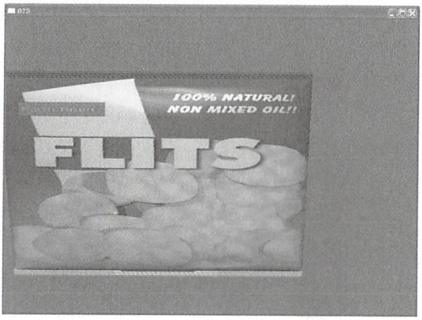
次のコードは、2,1,1の比率でスケーリングしています。YとZの縮尺はそのままで、X軸の縮尺のみ2倍にしています。その結果、横に伸びたレンダリングとなります。均等にスケーリングする場合は、3つのFLOAT値を全て同じにします。

▼コード片3

```
D3DXMATRIX matWorld;
D3DXMatrixScaling(&matWorld,2,1,1);
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
```

このコードを実際にサンプルの195行目あたりに打ち込んで、レンダリングを確認してください。また、スケーリング係数を変更して、その効果を確認してください。

▼コード片3のワールド変換を設定した場合



なお、ワールド行列の作成は簡単なものから複雑なものまで、要するに処理次第です。次のコードのように、中間計算の保持用に複数の行列を使うこともあります。というより、複数使うほうが一般的です。

```
D3DXMATRIX matWorld,matRotation;
     D3DXMatrixRotationY(&matWorld, timeGetTime()/3000.0f);
     D3DXMatrixRotationX(&matRotation, 0.5f);
     D3DXMatrixMultiply(&matWorld,&matWorld,&matRotation);
     pDevice->SetTransform(D3DTS_WORLD, &matWorld);
```

One Point

行列が難しいと思ってる人も多いですが、関数で行列を作成してくれるので、最初のうちは行列の意味を理解していなくても大丈夫です。

024 ビュー変換をする

シーン内のジオメトリをカメラ座標系に再配置する

初級

固有サンプルプロジェクト　024_ビュー変換の効果

3Dジオメトリのレンダリングに伴う3つの変換、ワールド変換、ビュー変換、プロジェクション変換の2番目の変換であるビュー変換はカメラ座標変換とも呼ばれます。

ワールド変換については、Tips023、プロジェクション変換についてはTips025を参照してください。

ビュー変換はワールド変換の次に行われる変換であり、その目的は、絶対座標系上に配置された3Dジオメトリを、さらにカメラ座標系上に配置することです。ワールド変換の目的はジオメトリの位置や姿勢をワールド座標上で反映させることですが、ビュー変換はジオメトリではなくカメラの位置等を反映させます。

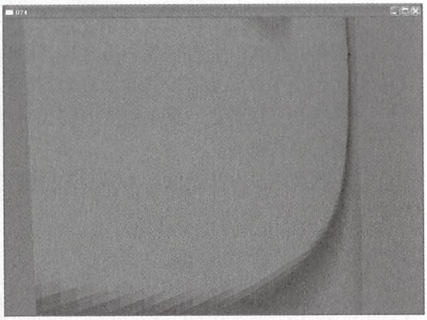
コード上、ビュー変換は次のようになります。

```
D3DXVECTOR3 vecEyePt; //カメラ（視点）位置
D3DXVECTOR3 vecLookatPt;//注視位置
D3DXVECTOR3 vecUpVec;//上方位置

D3DXMATRIX matView;
D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
pDevice->SetTransform(D3DTS_VIEW, &matView);
```

もちろん、ベクトル（vecEyePt等）や行列（matView）のインスタンス名は自由に命名できます。

ビュー変換のコードを書かないと、デフォルトでビュー変換されるので、サンプルでは右のようになってしまいます。

▼ビュー変換コードを書かない場合



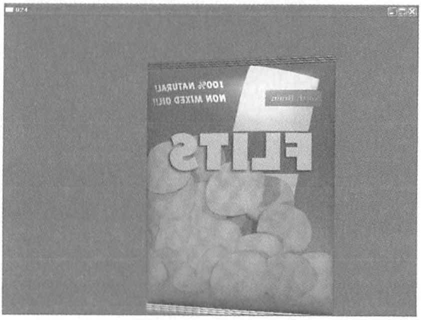
次のように、打ち込んでビルド、実行してみてください。

▼コード片1

```
D3DXMATRIX matView;

D3DXVECTOR3 vecEyePt(0.0f, 0.0f,-6.0f); //カメラ（視点）位置
D3DXVECTOR3 vecLookatPt(0.0f, 0.0f, 0.0f);//注視位置
D3DXVECTOR3 vecUpVec(0.0f, 01.0f, 0.0f);//上方位置

D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
pDevice->SetTransform(D3DTS_VIEW, &matView);
```

実行結果は、右のようになるはずです。

▼コード片1によるビュー変換を設定した場合



ビュー行列の作成は、D3DXMatrixLookAtLH関数により行います。
第1引数は、ビュー行列として用意した行列インスタンスのアドレス
第2引数は、視点ベクトルとして用意・初期化したベクトルインスタンスのアドレス、
第3引数は、注視点ベクトルとして用意・初期化したベクトルインスタンスのアドレス、
第4引数は、上方ベクトルとして用意・初期化したベクトルインスタンスのアドレス、
関数から戻ると、第1引数のビュー行列が初期化されます。

視点ベクトルとは、カメラの位置です。注視点とはカメラの焦点と言えばいいでしょうか。カメラの向きは視点から注視点に向かうベクトルとなります。

上方ベクトルは、「上はこの方向だよ」とDirect3Dに通知するためのものです。日常的な「上」はベクトルで表現すると(0,1,0)ですよね。3D空間では、「絶対的な上」という概念はないので、このように（日常的な上として）当たり前のことを教えてあげます。上方ベクトルを(0,-1,0)とするとレンダリング結果は上下反転するだけで、レンダリングそのものは正常に行われます。

One Point

ワールド変換がジオメトリ個々の変換なのに対し、ビュー変換は“カメラの位置や回転”を反映させます。基本的にシーン内にある全てのジオメトリはビュー変換の影響を受けます。
なお、右手座標系用にD3DXMatrixLookAtRH関数というのもあります。

025 プロジェクション変換をする

●カメラ座標系のジオメトリを2次元データに変換する

初級

固有サンプルプロジェクト　025_プロジェクション変換の効果

プロジェクション変換は、日本語に直すと透視変換です。透視や投影という言葉からなんとなく想像できるかもしれませんが、平たく言うと、「3次元データを“ぺらぺら”の2次元イメージにすること」とでも言えばいいでしょうか。

現実で格好の例えがあります。皆さんが紙に絵を書く時、立体である現実の3D物体を紙という2次元上に投影しているのは、プロジェクション変換に他なりません。ディスプレイは2次元の絵（ピクセルデータ）しか表示できませんから、プロジェクション変換が必要になります。

また、プロジェクション変換のパラメーターを調整することにより、現実のカメラ（のレンズ）のような効果も生まれます。

プロジェクション変換は、ワールド変換、ビュー変換と同じ手順を踏みます。

1. まず、D3DXMatrixPerspectiveFovLH関数によりプロジェクション変換に使用する行列を作成して、
2. IDirect3DDevice9::SetTransformメソッドにその行列を渡して実行します。第1引数のフラグはD3DTS_PROJECTIONです。

D3DXMatrixPerspectiveFovLHの第2引数は「視野角」を意味します。

視野角を狭くすると、物体が大きく写ると同時にパースがゆるくなります。ちょうど望遠レンズのようになります。逆に視野角を広くとると、物体が小さく写ると同時にパースがきつくなります。ちょうど広角レンズのようになります。

●視野角通常　標準レンズ

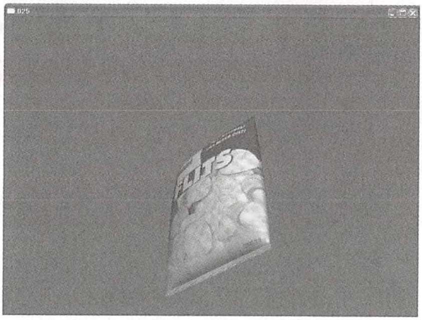
▼コード片1

```
D3DXVECTOR3 vecEyePt(0.0f, 0.0f,-8.0f);

D3DXMATRIXA16 matProj;
D3DXMatrixPerspectiveFovLH(&matProj,D3DX_PI/4,(FLOAT)WINDOW_WIDTH/(FLOAT)WIN
DOW_HEIGHT, 0.1f, 100.0f);
pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
```

標準の視野角の値は決まっているわけではありませんが、通常は45度くらいだと思います。3DCGの世界では、60分法ではなくラジアン法なので、45度はラジアンで指定します。

45度=180度÷4=π÷4（D3DX_PI/4）

▼通常の設定　コード片1



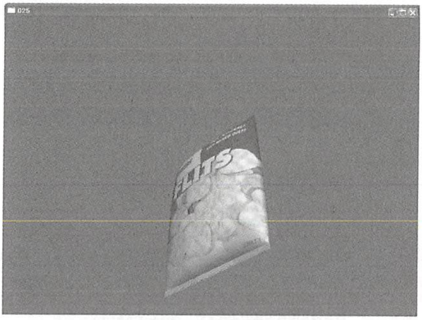
●視野角を狭くする　望遠レンズ

▼コード片2

```
D3DXVECTOR3 vecEyePt(0.0f, 0.0f,-125.0f);

D3DXMATRIXA16 matProj;
D3DXMatrixPerspectiveFovLH(&matProj,D3DX_PI/64,(FLOAT)WINDOW_WIDTH/(FLOAT)WINDOW_HEIGHT, 0.1f, 100.0f);
pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
```

効果がはっきり分かるようにかなりの望遠（視野角を2.8度（180/64度））にしました。パースがほとんどなくなっていますね。

▼望遠　コード片2



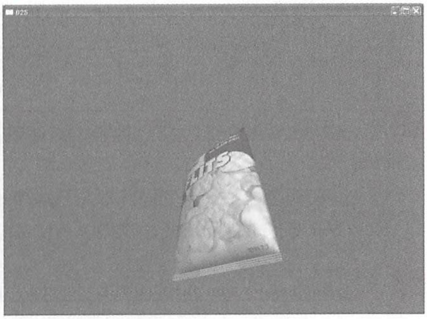
●視野角を大きくする　広角レンズ

▼コード片3

```
D3DXVECTOR3 vecEyePt(0.0f, 0.0f,-4.0f);

D3DXMATRIXA16 matProj;
D3DXMatrixPerspectiveFovLH(&matProj,D3DX_PI/2,(FLOAT)WINDOW_WIDTH/(FLOAT)WINDOW_HEIGHT, 0.1f, 100.0f);
pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
```

視野角度は90度です。かなりきついパースとなります。

▼ワイド　コード片3



One Point

プロジェクション変換は、コードを書いて明示的に設定しないと、レンダリング結果がなにも見えなくなってしまいます。

026 画面のフリップとは（画面更新をモニターに反映させる）

●IDirect3DDevice9::Presentメソッドにより画面を更新する　初級

ここで言う画面更新とは、バックバッファーとフロントバッファーを切り替えることです。これをフリップといいます。この切り替えを行わないと、ディスプレイには何も表示されません。

ただしフロントバッファーに直接絵をレンダリングした場合は、この切り替えは不要になります。ただ、フロントバッファーへの直接レンダリングは必ずテアリング（画面がちらつくこと）を起こしますので推奨されません。

通常はバックバッファーに一旦レンダリングして、その後でバックバッファーとフロントバッファーを切り替えてバックの絵を画面に引き出します。フロントバッファーとは、ディスプレイに現在表示されているピクセルのバッファーです。

したがって、フロントバッファーにアクセスして任意のピクセルを変更すると、その部分の変化が即座に画面に現われます。バックバッファーとは、フロントバッファーと同じピクセル格納バッファーであり、その存在意義はテアリング防止にあります。

なお、Direct3Dのフリップは、フロントとバックのそれぞれのポインターを切り替えることです。

つまり、バッファーの構造は全く同じであり、ある瞬間の機能による名前付けに過ぎま

せん。直前のフレームでは、フロントバッファーだったバッファーがバックバッファーになり、バックバッファーだったものがフロントバッファーとして機能します。

正式に書くと次のようになります。

```
IDirect3DDevice9::Present(NULL, NULL, NULL, NULL);
```

実際のコード上では、なんらかのインスタンスを作成するので、次のようになります。

```
pDevice->Present(NULL, NULL, NULL, NULL);
```

pDeviceはもちろんIDirect3DDevice9インターフェイスのポインターです。

One Point

バックバッファーは複数設けることができます。フロントバッファーと2つのバックバッファーがある時、それをトリプルバッファーと言い、通常のダブルバッファーよりも滑らかな（目に優しい）画面更新を行うことができます。

027 スプライトとは

●スプライト（妖精）は昔のゲームでは必須概念

初級

今も昔もスプライトの共通な定義は、2次元上の移動可能な絵を指します。そのような絵を描画する手段や実装方法は関係なく、あくまでもそのような絵の総称です。

スプライトという概念が生まれたのは、今から20数年前に遡ります。位置が固定されるテキストとは異なり、ディスプレイ上（スクリーン平面上）を自由に動き回る様子から、スプライト「妖精」と呼ばれ、多くのパソコンにはスプライトがハードウェアベースで実装されていました。当時のPCは処理速度が遅く、ピクセルやラインを組み合わせた汎用的なグラフィックス処理ではとても処理が追いつきません。ハード上にスプライト専用のユニットが存在し、汎用的なグラフィックとスプライトを別々に分けないと、まともな速度を得られなかったからです。

したがって、当初のスプライトの定義には、ハードウェアで実現するものという意味が含まれていました。このような専用ハードウェアベースのスプライトは当時のPCゲームにとって強力な武器だったので、ゲームに利用

される有名なPCには、テキスト、グラフィックのほかに、このスプライトという概念が当たり前のように実装されていました。

時代と共にPCのグラフィックス性能は向上し、現在はソフトベースで、他の3Dグラフィックパイプラインとともに画一的に描画できるまでになっています。それ故、もはやハードウェアにスプライト専用のユニットは不要となり、スプライトの定義も、その部分では微妙に変化しています。しかし、「2次元上の動く絵」という定義は同じであり、今後もその定義が変化することはないでしょう。

Direct3Dでは、2次元であるはずのスプライトを3次元上で処理しますが、それはスプライトの定義に反するわけではありません。なぜなら、冒頭で述べたように、その実現方法は関係ないからです。3次元ジオメトリを2次元平面に固定し、論理2Dとして振舞えば、それはスプライトです。

したがって、もしスプライトを斜めに傾けたり、奥行きを持たせたりすると、それはスプライトではなく、“ポリゴンに絵を貼った単なる3次元ジオメトリ”と呼ぶべきでしょう。

Direct3Dでは、スプライトを描画するためのスプライトインターフェイスが用意されています。また、そのようなインターフェイスを利用せずに自前でスプライトを作成することもよく行うことです。自分で3Dジオメトリに絵を貼り付けて論理2Dジオメトリとして描画すれば、それはスプライトです。

また、スプライトは「絵」ですから、論理2Dのポリゴンを論理2D上で動かしたからと言っても、それにテクスチャーが貼られていなければ、スプライトとは言えません。スプライトは「平面上で動く絵」ですから、この定義が当てはまるものは3次元上であったとしてもスプライトと呼べます。

028 スプライトにする画像の作成について

●スプライト画像を作成する際の注意点

初級

プログラム上でのスプライトオブジェクトやテクスチャ付きポリゴンの作成のことではなく、このTipsで述べるのは、スプライトに使用する“絵”そのものの作成についてであり、具体的に言うと、スプライトオブジェクトやポリゴンにテクスチャとして貼る何らかの画像ファイル（bmp、jpg、png等）のことです。

画像ファイルの作成やその他素材の作成段階の話になると、本書の範疇ではありませんが、参考までに述べておこうと思います。

スプライトに使用する画像ファイルは、何らかのツール（画像作成ソフト）で作成するか、素材集などから持ってくるか、何れにしても事前に用意しなくてはなりません（本書のサンプルでは素材は全て用意されています）。

絵心のある方であれば、Illustratorに代表されるドロー系ソフトで絵を書くのもいいでしょう。Photoshopのようなペイント系ソフトにより、撮影した写真を使うものいいかもしれません。

筆者は、2次元の絵でも3DCGソフトを使用して作成します。Lightwave3Dという3DCGソフトで3次元オブジェクトをレンダリングし、それをスプライトのテクスチャと

します。3DCGソフトと言ってもレンダリング結果は当然2次元の絵ですから、絵はいくらでも作れます。一旦3Dオブジェクトを作ると、さまざまなアングルの絵ができるという生産性や再利用性も3DCGならではです。

もし、本格的にゲームクリエイターを目指すのなら、大御所系の3Dソフトは持っているべきだと思います。Lightwave3D、3DStudioMAX、MAYA、SoftImageが4大CGソフトと呼ばれています。どのメーカーのものであるかということは、本質的な問題ではありません。3DCGの基本理論や自分なりの応用はどのソフトでも一緒です。何れか1つを極めれば、どんなソフトでもすぐに利用できます。

ただ、もし“就職“をすることを考えると、業界で多く使われているメーカーのものを使うのが（若干）良いのかもしれません。今までの経緯を見ると、4大ソフトの普及率は流動的です。どのソフトが一般的ということはありません。現在はMAYAが“人気“ですが、内部エンジンがパワフルなLightwaveが数年後のスタンダードになっているかもしれません。

何れにしても“ツールの使い方“などは半年もあればマスターできます。大切なのは、3DCG理論であって、ソフトのメーカーではありません。話の後半は脱線したような気がしますが、3DCGソフトが一本あると、スプライト作成に限らず、メッシュ、ムービー、そして当然3DCGの研究にも使えるということです。

ID3DXSpriteインターフェイスによるスプライト

029　スプライト（及びテクスチャ）オブジェクトを作成する

D3DXCreateSprite関数によりスプライトオブジェクトを作成する　初級

参考プロジェクト　030_スプライトをレンダリングする

スプライトをレンダリングするには、スプライトオブジェクトを作成する必要があります。

ただし、自前にポリゴンを作成し、それに絵を貼ることでスプライトを作成する場合はスプライトオブジェクトは不要です。それについてはTips046で解説しています。

スプライトオブジェクトを作成するには、D3DXCreateSprite関数を実行しますが、レンダリング時には初期化済みのテクスチャオブジェクトのアドレスが必要になります。つまり、スプライトオブジェクトをレンダリングするには、その前にテクスチャオブジェクトを作成しなければならないということです。

テクスチャオブジェクトの作成については、Tips140で詳細に解説していますので、ここでは、初期化例のみを記します。

スプライト自体の作成は、D3DXCreateSprite関数により行います。

第1引数、IDirect3DDevice9インターフェイスのポインター

第2引数、スプライトインターフェイスの（初期化前の）ポインター

関数が成功すると、第2引数が初期化されて、機能するようになります。

▼実際のコード片

```
LPDIRECT3DDEVICE9 pDevice;
LPDIRECT3DTEXTURE9 pTexture;
LPD3DXSPRITE pSprite;

//「テクスチャオブジェクト」の作成
D3DXCreateTextureFromFileEx(pDevice,"Sprite.bmp",SPRITE_WIDTH,
SPRITE_HEIGHT,0,0,D3DFMT_UNKNOWN,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAU
LT,
0xff000000,NULL,NULL,&pTexture);

// 「スプライトオブジェクト」の作成
D3DXCreateSprite(pDevice,&pSprite);
```

One Point

スプライトを自前で作成するのもいいですが、スプライトインターフェイスは“スプライト（スクリーン平面上の絵）の実装に便利です。

030 スクリーン平面上にレンダリングする

完全な2次元オブジェクトとして、スクリーン“平面”にレンダリングする

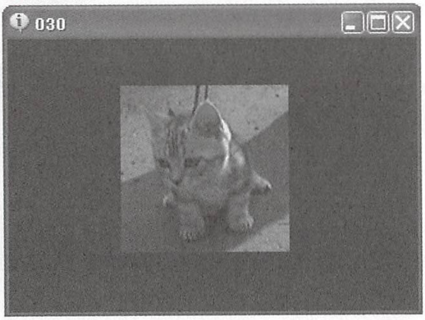
初級

固有サンプルプロジェクト　030_スクリーン平面上にレンダリングする

「スプライトをレンダリングするには？」という問いに対して例示するとすれば、本Tipsの固有サンプル全体になりますが、レンダリングに関係している部分は次のコード片です。これはサンプル内のDrawSprite関数の中身全てでもあります。

画面のクリアとBeginSceneとEndScene及び、Presentのコールがすでに存在するコードに埋め込む場合は、当然それらを除いた部分を書くことになります。

▼実行結果



```
pDevice->Clear(0, NULL, D3DCLEAR_TARGET,D3DCOLOR_XRGB(30,30,30), 1.0f, 0);
if(SUCCEEDED(pDevice->BeginScene()))
{
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(0,0,0);
    D3DXVECTOR3 vec3Position(350,250,0);
    pSprite->Begin(D3DXSPRITE_ALPHABLEND);
    pSprite->Draw(pTexture,&rect,&vec3Center,&vec3Position,D3DCOLOR_ARGB(255,255,255,255));
    pSprite->End();
    pDevice->EndScene();
}
pDevice->Present(NULL, NULL, NULL, NULL);
```

スプライト固有の部分は、pSprite->Begin からpSprite->Endまでの3行です。Drawメソッドに渡す引数の意味は、次のとおりです。

第1引数　スプライトに貼るテクスチャのポインター

第2引数　テクスチャ内の領域を指定する矩形のポインター。NULLを指定するとテクスチャ全体が貼られます。

第3引数　スプライトの中心を指定します。NULLを指定することとベクトル(0,0,0)を渡すことは同じことです。(0,0,0)を指定するとスプライトの左上端を中心に回転してしまうので、中心ベクトル(横幅の半分、縦幅の半分,0）を渡すと自然な回転をするようになります。

第4引数　スプライトのレンダリング位置をベクトルで指定します。座標はスクリーン座標なので簡単に指定できます。たとえば800x600のウィンドウで(400,300,0)と指定すればウィンドウの真ん中にレンダリングされます。

第5引数　色とアルファ値です。サンプルのようにD3DCOLOR_ARGBマクロを利用すると直感的に指定できます。

One Point

スプライトは、スクリーン座標なので、難しい3Dの計算は不要です。

Column

ツールの種類は関係ない

もし、筆者がMAYA専門の広告代理店の採用担当者だとして、採用選考中のA君とB君がいるとします。

A君はMAYAを、目を閉じても自由に操作できるほどに慣れています。そして3DCG理論の知識も人並みに持っています。

一方のB君はLightwaveしか使ったことがなく、MAYAには触れたこともありません。しかしB君は、3DCG理論及び現実のカメラ理論に卓越していない限り不可能なライティングで、驚くほどリアルな絵をレンダリングする技術を持っています。

筆者は土下座してでもB君を採用します。

031 スクリーン平面上で平行移動する

●表示座標を変更して、スクリーン平面上を平行移動する

初級

サンプルプロジェクト　031 スクリーン平面上で平行移動

ID3DXSpriteを2Dスプライトとして使用する場合において全般的に言えますが、その扱いは非常に簡単です。特に平行移動は最も簡単で直感的でしょう。

Drawメソッドの第4引数に座標を渡せば、それに見合ったスクリーン座標にレンダリングするようになります。座標はピクセル単位なので、何も難しいことを考える必要はありません。

サンプルでは、まずグローバルに座標用のベクトルを用意しています。

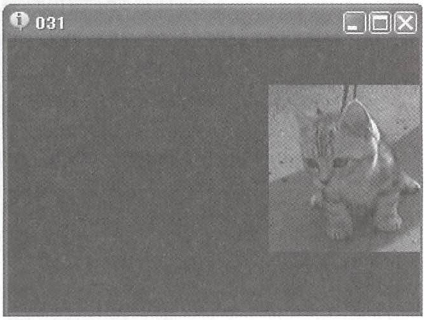
D3DXVECTOR3 g_vecPosition(150,100,0);

これは横150ピクセル、縦100ピクセルの座標という意味であり、スプライトは最初にその座標にレンダリングされることになります。

そして、ウィドウプロシージャー関数内でユーザーのキー操作に応じて、その座標を増減しています。

4つの矢印キーで上下左右に移動するようにしました。

▼実行結果



```
case WM_KEYDOWN:
    switch((CHAR)wParam)
    {
        case VK_ESCAPE:
            PostQuitMessage(0);
            break;
        case VK_LEFT:
            g_vecPosition.x-=2;
            break;
        case VK_RIGHT:
            g_vecPosition.x+=2;
            break;
        case VK_UP:
            g_vecPosition.y-=2;
            break;
```

```
           case VK_DOWN:
                 g_vecPosition.y+=2;
                 break;
    }
    break;
```

あとは、Drawメソッドの第4引数に、その座標を渡せばいいだけです。

```
pDevice->Clear(0, NULL, D3DCLEAR_TARGET,D3DCOLOR_XRGB(30,30,30), 1.0f, 0);
    if(SUCCEEDED(pDevice->BeginScene()))
    {
           RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
           D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);
           pSprite->Begin(D3DXSPRITE_ALPHABLEND);
           pSprite->Draw(pTexture,&rect,&vec3Center,&g_vecPosition,
D3DCOLOR_ARGB(255,255,255,255));
           pSprite->End();
           pDevice->EndScene();
    }
    pDevice->Present(NULL, NULL, NULL, NULL);
```

One Point

座標は、ピクセル単位であることに留意してください。

032 スクリーン平面上で回転させる

回転行列を渡して、スプライトを回転させる

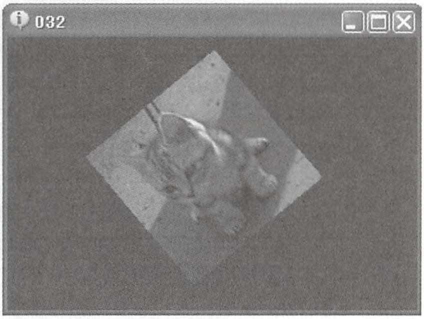
初級

サンプルプロジェクト 032 スクリーン平面上で回転

平行移動と異なり、回転は、回転行列を作成しなければなりません。しかし、行列作成用の関数が用意されているので、難しいことはありません。

サンプルでは2つの矢印キーで時計周り、反時計回りに回転するようにしました。回転行列の作成はD3DXMatrixRotationZ関数により行います。回転と同時に移動行列も作成し、その2つの行列を合成したワールド行列をSetTransformメソッドに渡します。

▼実行結果



```
D3DXMATRIX matWorld,matRotation,matTranslation;
D3DXMatrixRotationZ(&matRotation,g_fRoll);
D3DXMatrixTranslation(&matTranslation,g_vecPosition.x,g_vecPosition.y,0);
matWorld=matRotation * matTranslation;
pSprite->SetTransform(&matWorld);
```

これで、回転と移動を反映するようになります。

Tips031では、第4引数に直接、座標を渡しましたが、ここでは平行移動の変換を含むワールド行列を設定するスタイルなので、第4引数に座標を渡さなくても、位置を指定することができます。

One Point

スプライトの位置変換は、第4引数に渡す方法と、行列により3次元変換する方法の2つがあります。

033 スクリーン平面上で拡大縮小する

●スケーリング行列を作成して、スプライトを拡大縮小します　初級

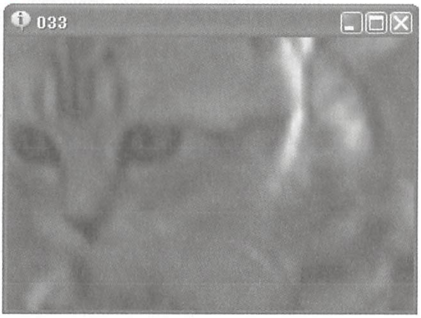
サンプルプロジェクト　033 スクリーン平面上で拡大縮小92

平行移動と異なり、拡大縮小（スケーリング）は、スケーリング行列を作成しなければなりません。しかし、行列作成用の関数が用意されているので、難しいことはありません。

サンプルでは、左右矢印キー2つで、拡大と縮小をするようにしました。

スケーリング回転行列の作成はD3DXMatrixScaling関数により行います。スケーリングと同時に移動行列も作成し、その2つの行列を合成したワールド行列をSetTransformメソッドに渡します。

▼実行結果



```
D3DXMATRIX matWorld,matScale,matTranslation;
D3DXMatrixScaling(&matScale,g_fScale,g_fScale,g_fScale);
D3DXMatrixTranslation(&matTranslation,g_vecPosition.x,g_vecPosition.y,0);
matWorld=matScale * matTranslation;
pSprite->SetTransform(&matWorld);
```

これで、スケーリングと移動を反映するようになります。

Tips031では、第4引数に直接、座標を渡しましたが、ここでは平行移動の変換を含むワールド行列を設定するスタイルなので、第4引数に座標を渡さなくても、位置を指定することができます。

One Point

回転とスケーリングは、行列を使用しなくてはなりません。

034 スクリーン平面上で半透明にする

●アルファ値を変更して、半透明にレンダリングする

初級

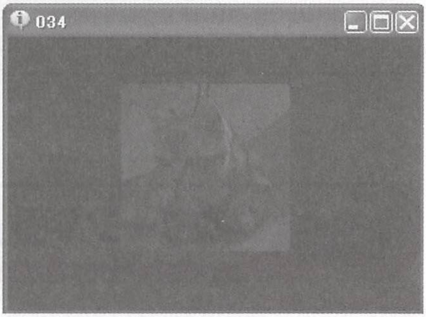
サンプルプロジェクト 034 スクリーン平面上で半透明

サンプルでは、左右の矢印キーで、透明度を上げ下げするようにしました。

透明度の設定は非常に簡単です。Drawメソッドの第5引数に渡すカラーの透明度を変更すればいいだけです。分かり易いようにD3DCOLOR_ARGBマクロを使用しています。

D3DCOLOR_ARGBマクロの4つの値のうち、最初の値がアルファブレンドの係数、すなわち、透明度を表します。

▼実行結果



```
pSprite->Draw(pTexture,&rect,&vec3Center,&g_vecPosition,D3DCOLOR_ARGB
(g_bAlpha,255,255,255));
```

サンプルでは、グローバルに用意したg_bAlphaとう変数をそのまま渡しています。g_bAlphaは、ウィンドウプロシージャー関数内で、ユーザーのキー入力に応じて次のように更新しています。

```
case VK_LEFT:
    g_bAlpha>0 ? g_bAlpha-=2 : NULL;
    break;
case VK_RIGHT:
    g_bAlpha<255 ? g_bAlpha+=2 : NULL;
    break;
```

次はレンダリング部分のコードです。

```
RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);
```

```
pSprite->Begin(D3DXSPRITE_ALPHABLEND);

pSprite->Draw(pTexture,&rect,&vec3Center,&g_vecPosition,D3DCOLOR_ARGB
(g_bAlpha,255,255,255));

pSprite->End();
```

One Point

D3DCOLOR_ARGBマクロを使用しない場合は、直接に32ビット値を渡します。例えば0xffxxxxxx(不透明)、0x77xxxxxx(半透明)、0x11xxxxxx(ほとんど透明)など。

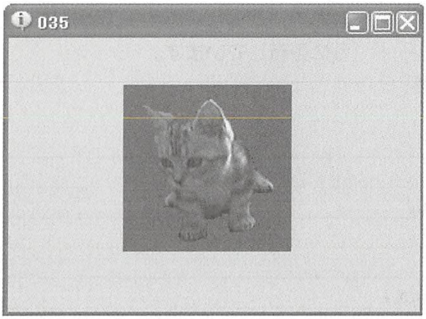
035 スクリーン平面上で“ふち”を背景色に馴染ませる

●テクスチャのカラーキーを指定して、特定の色を抜く

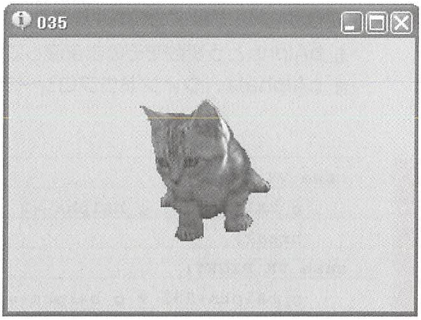
初級

サンプルプロジェクト　035 スクリーン平面上での“ふち”処理

▼実行結果（35-1）



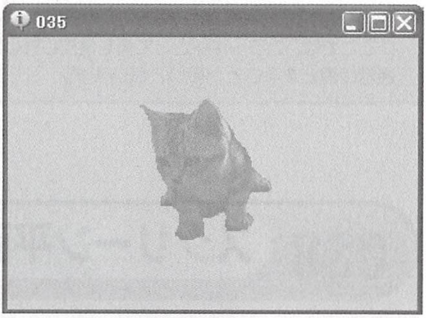
▼実行結果（35-2）



“ふち”とは、絵の周囲を取り囲む色のことです。図35-1では猫が絵であり、それ以外の領域を“ふち”と表現します。

通常、なにも意識しないでスプライトをレンダリングすると図35-1のようになります。“ふち”もそのままレンダリングされてしまい、“四角形の絵”ということが如実に見えてしまいます。“ふち”を見えなくすることは、つまり、“ふち”の部分だけ完全透明にするわけです。すると、図35-2のように、“絵”だけが見えるようにレンダリングすることができます。

▼実行結果（35-3）



```
D3DXCreateTextureFromFileEx(pDevice,"Sprite_blueback.bmp",SPRITE_WIDTH,SPRIT
E_HEIGHT,0,0,D3DFMT_UNKNOWN,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,0x
ff0000ff,NULL,NULL,&pTexture)))
```

第11引数は、「カラーキー」と呼ばれる色値です。カラーキーに指定した色の部分は、レンダリングされず、ちょうどその色だけを“抜いた”格好になるので「抜け色」とも呼ばれます。カラーキーにより、色を抜くことをカラーキーイングと言います。

カラーキーは、おもにDirectX7時代のDirectDrawで盛んに使用された技術ですが、主に2Dにおけるピクセルレベルで使用する技術なのでDirect3Dでは、テクスチャ及びスプライトでのみ使用することになります。

ここで使用している猫のビットマップは、ふちの色が青（rgb:0000ff）です。その青を抜きたいので、カラーキーを0xff0000ffとしています。もし、ふちの色が赤(rgb:ff0000)であれば、カラーキーを0xffff0000と指定することになります。

カラーキーに指定できる色は、たった一つだけなので、色抜きしたい場合、ふちは“単色”にしなくてはなりません。また、カラーキーは、正確に指定しないと上手く抜けません。

カラーキーに0xff0000ffと完全な青を指定してもビットマップ側のふちの色が部妙にずれている場合、たとえば0000feとかである場合は全く抜けません。そのような理由から、カラーキーイングする際にjpegなどの圧縮画像は不向きです。圧縮画像は、異なる色のピクセル境界を中間色にしてしまうため完全に単色にすることが難しく、汚い輪郭のようなピクセルが残ってしまいます。

カラーキーとスプライト自体のアルファブレンドは別物です。スプライトの透明度はDrawメソッドで指定します。サンプルでは、絵の部分の透明度を矢印キーで変更できるようにしました。

```
pSprite->Draw(pTexture,&rect,&vec3Center,&g_vecPosition,D3DCOLOR_ARGB
(g_bAlpha,255,255,255));
```

One Point

ここでは、カラーキーとして青を使用しましたが、通常は黒とすることが多いと思います。

036 スクリーン平面上で意図した重ね順で重ねる

●BeginメソッドにD3DXSPRITE_SORT_DEPTH_FRONTTOBACKフラグを加える

初級

サンプルプロジェクト　036スクリーン平面上で重ねる

スプライトオブジェクトを2Dスプライトとして運用している場合は、第4引数の3次元ベクトルのZ成分は、「重ね順の手掛かり」くらいの意味しか持たず、Z成分を増減させたからといって、スプライトの奥行きが変化するわけではありません。

他方、スプライトオブジェクトを3次元ジオメトリのビルボードとして運用する場合は、Z成分の変更が奥行きに反映されます。その辺については、ワールド空間でのスプライト関連であるTips037～Tips044を参照してください。

スクリーン平面上で、意図した順番で重ねるには、2つのアプローチがあります。

1つ目は、奥にレンダリングしたいものは先にレンダリングするというルールを守ることです。

2つ目は、ソートフラグ及び、ソートのもととなるZ成分を指定する方法です。

ここでは2つ目のアプローチをとります。1つ目のアプローチは、レンダリングのタイミングを気にしなければならず、現実的にはなかなか難しいでしょう。

ただ、1つ目のルールは、2次元上へのレンダリングの大原則ではあります。そもそも、スクリーン“平面”なのですから、重なる（ように見える）ためには、先にレンダリングしたスプライトの上に上書きするしかありません。言い換えれば、先にレンダリングした絵の裏側にレンダリングすることは困難です。

この問題を解決するために、Direct3Dのスプライトは、Drawメソッドを実行した瞬間にレンダリングされるのではなく、一旦バッチリストというリストに登録します。

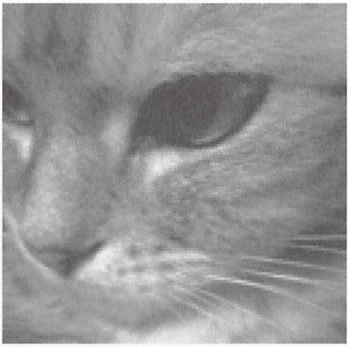
実際にレンダリングされるのは、Endメソッドをコールした時にリストにあるすべてのスプライトがレンダリングされます。つまり、Drawメソッドを実行した分だけ、どんどん描画リストにたまっていき、Endメソッドにより一気にレンダリングするというものです。

この仕組みにより、レンダリングのタイミングに関らず重ね順をソートすることが可能となり、レンダリングした順番に依存しない意図した順番で重ねることができます。

サンプルでは、次の2枚のスプライトを用意しました。

▼スプライト36-1

▼スプライト36-2



Z成分とソートフラグを指定すれば、この2枚をレンダリングの順番とは関係なく、重ねることができます。ソートフラグはD3DXSPRITE_SORT_DEPTH_FRONTTOBACKを使用します。このフラグの意味は、Z成分による奥行きが後ろスプライトは背面にレンダリングされるように並び替えるということです。

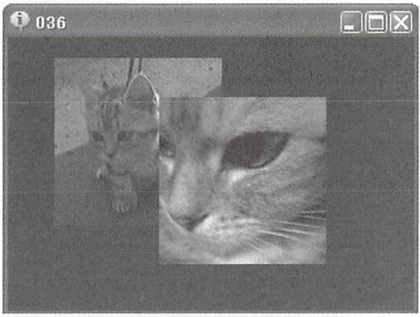
```
pSprite->Begin(D3DXSPRITE_ALPHABLEND | D3DXSPRITE_SORT_DEPTH_FRONTTOBACK);
```

このフラグによりBeginすれば、Z成分に見合った重なり順でレンダリングできるようになります。

●1枚目のZ成分を1（奥）、2枚目のZ成分を0にした場合

```
pSprite->Begin(D3DXSPRITE_ALPHABLEND | D3DXSPRITE_SORT_DEPTH_FRONTTOBACK);
//1枚目のスプライト
D3DXVECTOR3 vec3Position(100,80,1);
pSprite->Draw(pTexture,&rect,&vec3Center,&vec3Position,D3DCOLOR_ARGB
(255,255,255,255));
//2枚目のスプライト
D3DXVECTOR3 vec3Position_2(180,110,0);
pSprite->Draw(pTexture_2,&rect,&vec3Center,&vec3Position_2,D3DCOLOR_ARGB
(255,255,255,255));
pSprite->End();
```

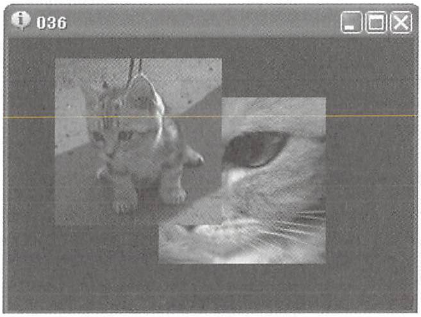
▼実行結果



●1枚目のZ成分を0、2枚目のZ成分を1（奥）にした場合

```
//1枚目のスプライト
D3DXVECTOR3 vec3Position(100,80,0);
pSprite->Draw(pTexture,&rect,&vec3Center,&vec3Position,D3DCOLOR_ARGB
(255,255,255,255));
//2枚目のスプライト
D3DXVECTOR3 vec3Position_2(180,110,1);
pSprite->Draw(pTexture_2,&rect,&vec3Center,&vec3Position_2,D3DCOLOR_ARGB
(255,255,255,255));
pSprite->End();
```

▼実行結果



One Point

Drawメソッドを実行した時点では、スプライトはバッチリストに登録されるだけ！

バッチリストにより、重ね順のソートを生かすことができます。

037 ワールド空間内にレンダリングする

D3DXSPRITE_BILLBOARDフラグでBeginする

中級

サンプルプロジェクト　037ワールド空間内にレンダリングする

キャプチャ絵の見かけは、2Dスプライトと同じですが、これは3次元空間上にあるスプライトのレンダリング結果です。というよりも、3次元空間内であっても“スプライト”をレンダリングしようとしているのですから、見かけが2Dスプライトと同じでなくてはスプライトの定義に反します（スプライトとは呼べなくなってしまいます）。

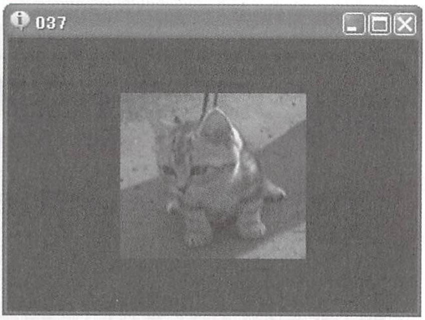
ID3DXSpriteであろうが、テクスチャードクアッドであろうが、3次元ジオメトリを“スプライト”とするためには、「ビルボーディング」というテクニックを適用する必要があります。

ビルボーディングをかけた物体は、常にカメラをのほうを向くので、レンダリング上は常に正面を向きます。テクスチャ付きポリゴンがそのような状態であれば、“スプライト”と呼べます。

ID3DXSpriteを利用する場合は、ビルボーディングをコーディングできなくても構いません。BeginメソッドにD3DXSPRITE_BILLBOARDフラグを指定すれば、インターフェイス側でビルボーディングをかけてくれます。ただし、ビルボーディングの計算に必要な現在のワールド行列とビュー行列はID3DXSprite::SetWorldViewLHメソッドにより渡してやらなければなりません。

次のコード片はサンプルからの抜粋ですが、レンダリング部分のコーディングはだいたいこのようになります。

▼実行結果



```
if(SUCCEEDED(pDevice->BeginScene()))
{
    //ID3DXSpriteによるスプライトのレンダリング
    {
        D3DXMATRIX matWorld,matTrans;
        RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
        D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);//中心はスプライト画像の真ん中
        D3DXMatrixTranslation(&matTrans,0,0,128);//128単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。(巨大サイズになるので)
```

```
          matWorld=matTrans;
          pSprite->SetWorldViewLH(&matWorld,&matView);
          pSprite->SetTransform(&matWorld);
          pSprite->Begin(D3DXSPRITE_BILLBOARD);//ビルボードフラグにするとスプライ
トは３Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB
(255,255,255,255));
          pSprite->End();    //バッチスプライト（描画リストに溜まっている全スプライト）
を描画。もっとも本サンプルは一個だけだが
    }
    pDevice->EndScene();
}
```

One Point

草木や爆発の煙など、複数のスプライトを空間的に配置したい場合は、ここでのように3次元空間にレンダリングする必要があります。

3次元空間内で運用する場合は、座標はピクセル単位ではないことに留意してください。

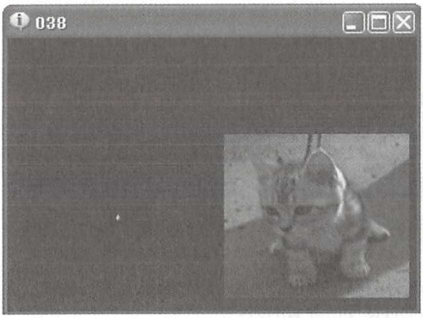
038 ワールド空間内に平行移動する

移動行列を基にワールド変換を行うことによりスプライトを平行移動する

中級

サンプルプロジェクト　038 ワールド空間内で平行移動

▼実行結果



平行移動用の行列を用意して、D3DXMatrixTranslation関数により平行移動行列として初期化します。

D3DXMatrixTranslation関数は、指定されたX,Y,Z3つの座標成分から、平行移動行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した平行移動行列をワールド変換行列に含めて（サンプルではほかの変換がないので、そのままワールド行列とも言えます）、そのワールド行列をID3DXSprite::SetTransformメソッドによりワールド変換として登録します。

なお、ビルボーディングの計算に必要な現在のワールド行列とビュー行列はID3DXSprite:: SetWorldViewLHメソッドにより渡してやらなければなりません。

```
//ID3DXSpriteによるスプライトのレンダリング
{
    D3DXMATRIX matWorld,matTrans;
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);  //中心はスプライト画像の真ん中
    D3DXMatrixTranslation(&matTrans,g_vecPosition.x,g_vecPosition.y,g_vecPosition.z+128);//128単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。(巨大サイズになるので)
    matWorld=matTrans;
    pSprite->SetWorldViewLH(&matWorld,&matView);
    pSprite->SetTransform(&matWorld);
    pSprite->Begin(D3DXSPRITE_BILLBOARD);//ビルボードフラグにするとスプライトは3Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB(255,255,255,255));
    pSprite->End();      //バッチスプライト（描画リストに溜まっている全スプライト）を描画。もっとも本サンプルは一個だけだが
}
```

One Point

3次元空間内で運用する場合は、座標はピクセル単位ではないことに留意してください。

第4引数に座標を渡して配置する場合は、特に注意してください。ワールド変換と第4引数の移動が加算されて配置されます。

039 ワールド空間内で回転させる

回転行列を基にワールド変換を行うことによりスプライトを回転する

中級

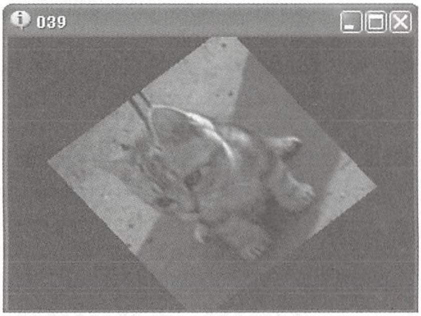
サンプルプロジェクト　039 ワールド空間内で回転

回転行列用の行列を用意して、D3DXMatrixRotationZ関数により回転行列として初期化します。
D3DXMatrixRotationZ関数は、指定された角度（ラジアン角）から、Z軸周りの回転行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した回転行列をワールド変換行列に含めて（サンプルではワールド変換は、回転と平行移動の合成変換です）、そのワールド行列をID3DXSprite::SetTransformメソッドによりワールド変換として登録します。

回転させる場合は、ビルボードフラグでは上手くいきません。D3DXSPRITE_OBJECTSPACEフラグによりBeginする必要があります。オブジェクトスペースの場合は、ビルボード計算に必要なワールド、ビュー行列を渡す必要がありませんので、ID3DXSprite:: SetWorldViewLHメソッドの実行は不要です。

▼実行結果



```
//ID3DXSpriteによるスプライトのレンダリング
{
    D3DXMATRIX matWorld,matRotation,matTrans;
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);  //中心はスプライト画像の真ん中
    D3DXMatrixRotationZ(&matRotation,g_fRoll);
    D3DXMatrixTranslation(&matTrans,0,0,200);//200単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。(巨大サイズになるので)
    matWorld=matRotation*matTrans;

    pSprite->SetTransform(&matWorld);
    pSprite->Begin(D3DXSPRITE_OBJECTSPACE);//オブジェクトスペースフラグにするとスプライトは3Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB(255,255,255,255));
    pSprite->End();     //バッチスプライト（描画リストに溜まっている全スプライト）を描画。
もっとも本サンプルは一個だけだが
}
```

One Point

ここではZ軸回りで回転させましたが、カメラの方向によってはX軸、Y軸で回転させてる場合もあるでしょう。その場合、D3DXMatrixRotationX、あるいはD3DXMatrixRotationY関数により回転行列を作成することになります。

040 ワールド空間内で拡大縮小する

●スケーリング行列を基にワールド変換を行うことによりスプライトをスケーリングする　中級

サンプルプロジェクト　040 ワールド空間内で拡大縮小

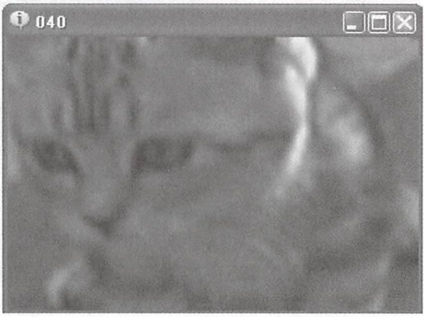
スケーリング行列用の行列を用意して、D3DXMatrixScaling関数によりスケーリング行列として初期化します。

D3DXMatrixScaling関数は、指定された3つの比率から、スケーリング行列を作成し、第1引数に指定した行列インスタンスに代入します。3つの比率は左から順にX,Y,Z軸方向における縮尺です。

作成したスケーリング行列をワールド変換行列に含めて（サンプルでのワールド変換は、スケーリングと平行移動の合成変換です）、そのワールド行列をID3DXSprite::SetTransformメソッドによりワールド変換として登録します。

なお、ビルボーディングの計算に必要な現在のワールド行列とビュー行列はID3DXSprite:: SetWorldViewLHメソッドにより渡してやらなければなりません。

▼実行結果



```
//ID3DXSpriteによるスプライトのレンダリング
{
    D3DXMATRIX matWorld,matScale,matTrans;
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);  //中心はスプライト画像の真ん中
    D3DXMatrixScaling(&matScale,g_fScale,g_fScale,g_fScale);
    D3DXMatrixTranslation(&matTrans,0,0,128);//128単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。（巨大サイズになるので）
    matWorld=matScale * matTrans;
    pSprite->SetWorldViewLH(&matWorld,&matView);
    pSprite->SetTransform(&matWorld);
    pSprite->Begin(D3DXSPRITE_BILLBOARD);//ビルボードフラグにするとスプライトは3Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB(255,255,255,255));
    pSprite->End();      //バッチスプライト（描画リストに溜まっている全スプライト）を描画。もっとも本サンプルは一個だけだが
}
```

One Point

3つの成分を均等に増減させると均等にスケーリングします。つまり、形の比率はそのままで拡大縮小します。3成分をばらばらに増減させてスケーリングした場合は、縦と横の何れかに歪むのでストレッチと言ったほうが良いかも知れません。

041 ワールド空間内で半透明にする

●Drawメソッドに渡すアルファ値を変更して、半透明にレンダリングする

中級

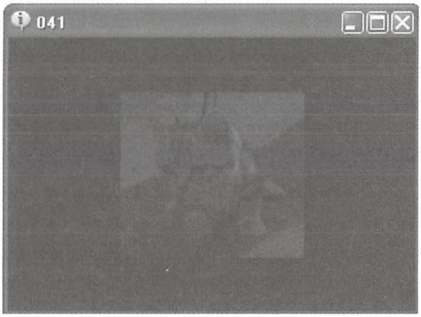
サンプルプロジェクト　041 ワールド空間内で半透明

サンプルでは、左右の矢印キーで、透明度を上げ下げするようにしました。

透明度の設定は非常に簡単です。Drawメソッドの第5引数に渡すカラーの透明度を変更すればいいだけです。分かり易いようにD3DCOLOR_ARGBマクロを使用しています。

D3DCOLOR_ARGBマクロの4つの値のうち、最初の値がアルファブレンドの係数、すなわち、透明度を表します。

▼実行結果



```
pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB
(g_bAlpha,255,255,255));
```

サンプルでは、グローバルに用意したg_bAlphaとう変数をそのまま渡しています。g_bAlphaは、ウィンドウプロシージャー関数内で、ユーザーのキー入力に応じて次のように更新しています。

```
case VK_LEFT:
    g_bAlpha>0 ? g_bAlpha-=2 : NULL;
    break;
case VK_RIGHT:
    g_bAlpha<255 ? g_bAlpha+=2 : NULL;
    break;
```

レンダリング部分のコードは次のとおりです。

```
//ID3DXSpriteによるスプライトのレンダリング
{
    D3DXMATRIX matWorld,matTrans;
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);  //中心はスプライ
ト画像の真ん中
    D3DXMatrixTranslation(&matTrans,0,0,128);//128単位奥に配置。3D空間内ではピク
セルがそのままサイズ単位になる。(巨大サイズになるので)
    matWorld=matTrans;
    pSprite->SetWorldViewLH(&matWorld,&matView);
    pSprite->SetTransform(&matWorld);
    pSprite->Begin(D3DXSPRITE_ALPHABLEND | D3DXSPRITE_BILLBOARD);//ビルボード
フラグにするとスプライトは3Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB
(g_bAlpha,255,255,255));
    pSprite->End();      //バッチスプライト（描画リストに溜まっている全スプライト）を描画。
もっとも本サンプルは一個だけだが
}
```

One Point

D3DCOLOR_ARGBマクロを使用しない場合は、直接に32ビット値を渡します。たとえば0xffxxxxxx（不透明）、0x77xxxxxx（半透明）、0x11xxxxxx（ほとんど透明）など。

042 ワールド空間内で“ふち”を背景色に馴染ませる

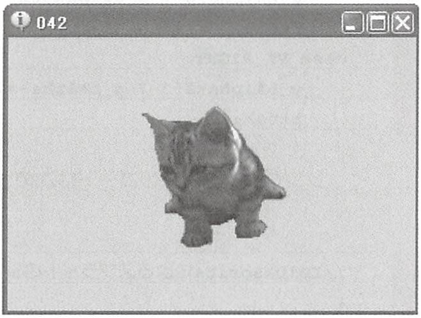
●テクスチャのカラーキーを指定して、特定の色を抜く

初級

サンプルプロジェクト　042 ワールド空間内での“ふち”処理

本質的に、Tips035と全く同じですので、Tips035を参照してください。

▼実行結果



```
    //「テクスチャオブジェクト」の作成
    if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite_Blueback_
Inverse.bmp",SPRITE_WIDTH*(16.0f/24.0f),SPRITE_HEIGHT*(16.0f/24.0f),
0,0,D3DFMT_UNKNOWN,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,0xff0000ff,
NULL,NULL,&pTexture)))
```

One Point

ふち処理は、テクスチャレベルの処理なのでワールド空間でも変わりはありません。

043 ワールド空間内で意図した重ね順で重ねる

BeginメソッドにD3DXSPRITE_SORT_DEPTH_FRONTTOBACKフラグを加える

中級

サンプルプロジェクト　043 ワールド空間内で重ねる

スクリーン平面での重ねと同様に、ワールド空間で重ねる場合も、D3DXSPRITE_SORT_DEPTH_FRONTTOBACKフラグを指定することにより位置的に奥にあるスプライトを背面にレンダリングします。

サンプルでは、まず両方のスプライトにワールド変換をかけています。これは、ワールド空間ではスプライトのサイズがピクセル分の単位サイズになるため、ピクセル幅（128）ぶんだけ後ろに押しやらないと大きすぎて何の絵か分からなくなるからです。

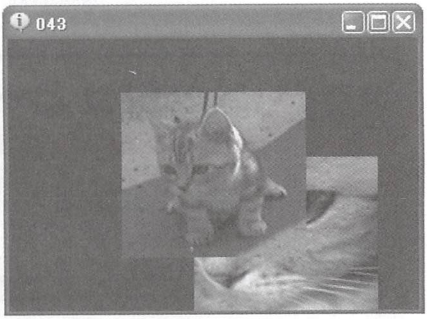
2枚とも同じワールド変換をかけるので、そのままでは同じ位置になります。そこで2枚目のスプライトに第4引数により1枚目と位置をずらしています。

128単位奥にあるので、50単位ずらしても、画面上はさほど動かないことに留意してください。

もし、1枚目のスプライトを背面にレンダリングしたい場合は、2枚目よりも後方の座標を第4引数に指定しましょう。

あるいは、別々のワールド変換を行うのであれば、第4引数はNULLでもいいでしょう。

▼実行結果



```
//ID3DXSpriteによるスプライトのレンダリング
{
    D3DXMATRIX matWorld,matTrans;
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);  //中心はスプライト画像の真ん中
    D3DXMatrixTranslation(&matTrans,0,0,128);//128単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。(巨大サイズになるので)
    matWorld=matTrans;
    pSprite->SetWorldViewLH(&matWorld,&matView);
    pSprite->SetTransform(&matWorld);
    pSprite->Begin(D3DXSPRITE_BILLBOARD | D3DXSPRITE_SORT_DEPTH_FRONTTOBACK);//ビルボードフラグにするとスプライトは3Dワールド空間で処理される
```

```
        //1枚目のスプライト
        pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB
(255,255,255,255));
        //2枚目のスプライト 第4引数のベクトルは、現在のワールド変換を基準とした相対的な移動量と
して機能する。
        pSprite->Draw(pTexture_2,&rect,&vec3Center,&D3DXVECTOR3(50,-
50,1),D3DCOLOR_ARGB(255,255,255,255));
        pSprite->End();       //バッチスプライト（描画リストに溜まっている全スプライト）を描画。
もっとも本サンプルは一個だけだが
}
```

One Point

通常、3D空間では1単位でもけっこうな大きさでレンダリングされますので、仮に128ピクセルという小さなスプライトでも128単位になるので、非常に大きなジオメトリに相当します。スプライトのサイズ分だけ後方に配置すると適度な大きさになります。

044 メッシュと一緒にレンダリングする

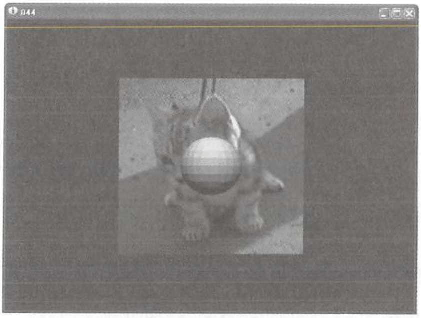
メッシュと同じく3Dジオメトリと捉えてレンダリングする

中級

サンプルプロジェクト　044 メッシュと一緒にレンダリング

ワールド空間でのID3DXSpriteは3Dジオメトリとして扱うので、なにも考えずにメッシュと一緒にレンダリングすることができます。サンプルでは、別々のワールド変換をかけて。メッシュのほうは回転するようにしました。

▼実行結果



```
if(SUCCEEDED(pDevice->BeginScene()))
{
        //メッシュのワールドトランスフォーム（絶対座標変換）
```

```
    D3DXMatrixRotationY(&matRotation, timeGetTime()/3000.0f);
    D3DXMatrixTranslation(&matTrans,0,0,5);
    matWorld=matRotation*matTrans;
    pDevice->SetTransform(D3DTS_WORLD, &matWorld);
    //メッシュのレンダリング
    for(DWORD i=0; i<dwNumMaterials; i++)
        {
            pDevice->SetMaterial(&pMeshMaterials[i]);
            pDevice->SetTexture(0, pMeshTextures[i]);
            pMesh->DrawSubset(i);
        }
    //ID3DXSpriteによるスプライトのレンダリング
    {
        RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
        D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);//中心はスプライト画像の真ん中
        D3DXMatrixTranslation(&matTrans,0,0,128);//128単位奥に配置。3D空間内ではピクセルがそのままサイズ単位になる。(巨大サイズになるので)
        matWorld=matTrans;
        pSprite->SetWorldViewLH(&matWorld,&matView);
        pSprite->SetTransform(&matWorld);
        pSprite->Begin(D3DXSPRITE_BILLBOARD);//ビルボードフラグにするとスプライトは3Dワールド空間で処理される
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB(255,255,255,255));
        pSprite->End();     //バッチスプライト（描画リストに溜まっている全スプライト）を描画。もっとも本サンプルは一個だけだが
    }
    pDevice->EndScene();
}
```

One Point

ワールド変換後の位置を基準としてDrawメソッドの第4引数により、さらに移動させせます。

045 フルスクリーンに復帰した時に再初期化をする

デバイスをリセットし、かつ、必要なオブジェクトを再度初期化する　中級

サンプルプロジェクト　045 フルスクリーンからの復帰

Alt+tabキーにより、画面をデスクトップに切り替えると、フルスクリーンでのリソースは破壊されます。破壊されたリソース（ここではスプライトとテクスチャオブジェクト）は元に戻さなくてはなりません。このことの本質については、Tips020を参照してください。

まずは、デバイスをリセットします。

```
//デバイス消失への対応
if(pDevice->Present(NULL, NULL, NULL, NULL)==D3DERR_DEVICELOST)
{
    if(pDevice->TestCooperativeLevel() == D3DERR_DEVICELOST)
    {
        return;
    }
    SAFE_RELEASE(pTexture);
    SAFE_RELEASE(pSprite);

    D3DPRESENT_PARAMETERS d3dpp;

    ZeroMemory(&d3dpp, sizeof(d3dpp));
    D3DDISPLAYMODE DMode;
    pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT ,&DMode);
    d3dpp.BackBufferFormat =DMode.Format;
    d3dpp.BackBufferWidth=WINDOW_WIDTH;
    d3dpp.BackBufferHeight=WINDOW_HEIGHT;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = FALSE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;

    if(pDevice->Reset(&d3dpp)!=D3D_OK)
    {
        MessageBox(0,"デバイスのリセットに失敗しました","",MB_OK);
    }
```

次にテクスチャを再初期化します。プログラム起動直後の初期化コードと全く同一です。

```
    //「テクスチャオブジェクト」の“再”作成
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite.jpg",SPRITE_WIDTH,SPRI
TE_HEIGHT,0,0,
D3DFMT_A8R8G8B8,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,
                0xff000000,NULL,NULL,&pTexture)))
    {
        MessageBox(0,"テクスチャの再構築に失敗しました","",MB_OK);
        return ;
    }
```

スプライトも再初期化します。これもプログラム起動直後の初期化コードと全く同一です。

```
    // 「スプライトオブジェクト」の“再”作成
    if(FAILED(D3DXCreateSprite(pDevice,&pSprite)))
    {
        MessageBox(0,"スプライトの再構築に失敗しました","",MB_OK);
        return ;
    }
}
```

One Point

要するに、デバイスをリセットしたあとは、初期化コードをそのままコピーするか、初期化ルーチンを再度呼び出せばいいわけです。

板ポリゴンで作成するスプライト（テクスチャードクアッド）

046 テクスチャードクアッドを作成する

●4頂点ポリゴンを作成し、そこにテクスチャを貼る

中級

参考プロジェクト　047 テクスチャードクアッドをレンダリングする

ID3DXSpriteを使用しなくても、スプライトを作れます。といよりも以前は誰しもこの方法のみでスプライトを作成していました。

まず最も単純な4頂点ポリゴンを作成します。ポリゴンは4角形にするのが一般的です。そこにテクスチャを貼り付けます。テクスチャの作成と貼り付け、ID3DXSpriteの時と同一です。このような“絵付き4角ポリゴン”はテクスチャードクアッド（Textured Quad）と呼ばれています。「絵を施した」（テクスチャード）「四角形」（クアッド）という意味です。そしてテクスチャードクアッドにビルボーディングをかけるとスプライトとして振舞うことができます。

ポリゴンの作成は簡単です。まず次のように型を定義します。

```
//頂点フォーマット定義
struct TEX_VERTEX
{
    FLOAT x,y,z;
    FLOAT u,v;
};
```

グローバルに、あるいは、ローカルに次のように初期化すればいいだけです。たった4頂点なのでローカルに初期化を行っても問題はありません。

```
TEX_VERTEX vecPoint[4+1]=
{
    { -0.5,0.5,0,0,0 }, //頂点1
    {  0.5,0.5,0,1,0 }, //頂点2
    { -0.5,-0.5,0,0,1 },//頂点3
    { 0.5,-0.5,0,1,1 }  //頂点4
};
```

ポリゴンに貼り付けるテクスチャは、ID3DXSpriteのTipsで解説した方法と全く同様にD3DXCreateTextureFromFileEx関数により行います。

```
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite.jpg",SPRITE_WIDTH＊
(16.0f /24.0f),SPRITE_HEIGHT＊(16.0f/24.0f),0,0,D3DFMT_UNKNOWN,
D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,0xff000000,NULL,NULL,&pTexture
)))
{
    MessageBox(0,"テクスチャの作成に失敗しました","",MB_OK);
    return E_FAIL;
}
```

あるいは、

```
struct TEX_VERTEX
{
        D3DXVECTOR3 Coord;
        FLOAT u,v;
};
```

とか、

```
struct TEX_VERTEX
{
        D3DXVECTOR3 Coord;
        D3DXVECTOR2 TexCoord;
};
```

などとしても構いません。

047 テクスチャードクアッドをレンダリングする

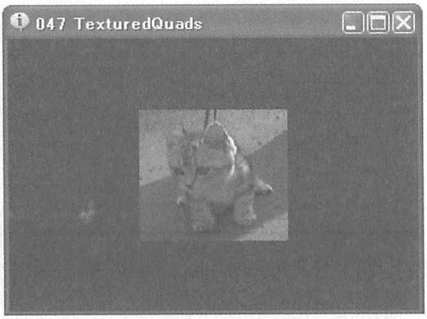
IDirect3DDevice9::DrawPrimitiveUPメソッドを使う　中級

サンプルプロジェクト　047 テクスチャードクアッドをレンダリングする

ここでは最もシンプルにポリゴンを作成できるDrawPrimitiveUPによりレンダリングします。
頂点の初期化は見やすさを優先して、ローカルに行っています。ローカルな初期化なので毎フレーム同じ初期化を行いますが、たった4頂点なので問題はありません。

頂点の配列を用意したら、その頂点の型をIDirect3DDevice9:: SetFVFメソッドによりセットします。次に、IDirect3DDevice9:: DrawPrimitiveUPメソッドによりレンダリングします。サンプルでは、その間にIDirect3DDevice9::SetTextureメソッドによりテクスチャをセットしていますが、テクスチャのセットはどこでも構いません。

▼実行結果



```
VOID Render()
{
    pDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB
(30,30,30), 1.0f, 0);
    if(SUCCEEDED(pDevice->BeginScene()))
    {
          //ワールドトランスフォーム（絶対座標変換）
          D3DXMATRIXA16 matWorld;
          D3DXMatrixTranslation(&matWorld,0,0,1);
          //D3DXMatrixRotationY(&matWorld, timeGetTime()/1000.0f);
          pDevice->SetTransform(D3DTS_WORLD, &matWorld);
          //テクスチャードクアッドのレンダリング
          {
                TEX_VERTEX vecPoint[4+1]=
                {
                      { -0.5,0.5,0,0,0 }, //頂点1
                      {  0.5,0.5,0,1,0 }, //頂点2
                      { -0.5,-0.5,0,0,1 },//頂点3
                      { 0.5,-0.5,0,1,1 }  //頂点4
                };

                pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
                pDevice->SetTexture(0,pTexture);
                pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint,
sizeof(TEX_VERTEX));
          }

          pDevice->EndScene();
    }
    pDevice->Present(NULL, NULL, NULL, NULL);
}
```

One Point

IDirect3DDevice9:: DrawPrimitiveUPメソッドは、頂点バッファの用意等、わずらわしい準備をせずにプリミティブ（ポリゴン、点、ライン）をレンダリングできるのでお手軽です。

048 テクスチャードクアッドを平行移動する

移動行列を基にワールド変換を行うことにより平行移動する

中級

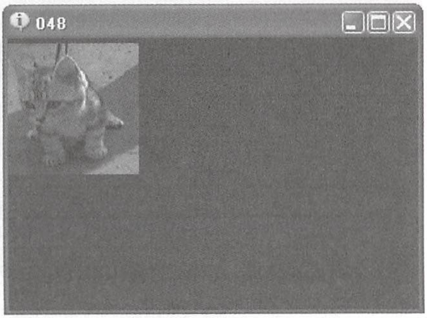
サンプルプロジェクト 048 テクスチャードクアッドを平行移動

平行移動用の行列を用意して、D3DXMatrixTranslation関数により平行移動行列として初期化します。

D3DXMatrixTranslation関数は、指定されたX,Y,Z3つの座標成分から、平行移動行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した平行移動行列をワールド変換行列に含めて（サンプルではほかの変換がないので、そのままワールド行列とも言えます）、そのワールド行列をIDirect3DDevice9::SetTransform(D3DTS_WORLD, OOO)メソッドによりワールド変換として登録します。

▼実行結果



```
//ワールドトランスフォーム（絶対座標変換）
D3DXMATRIXA16 matWorld;
D3DXMatrixTranslation(&matWorld,g_vecPosition.x,g_vecPosition.y,1);
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
//テクスチャードクアッドのレンダリング
{
    TEX_VERTEX vecPoint[4+1]=
    {
            { -0.5,0.5,0,0,0 }, //頂点1
            {  0.5,0.5,0,1,0 }, //頂点2
            { -0.5,-0.5,0,0,1 },//頂点3
            { 0.5,-0.5,0,1,1 }  //頂点4
    };

    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
    pDevice->SetTexture(0,pTexture);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));
}
```

One Point

テクスチャードクアッドは3次元ジオメトリですので、ピクセル単位での座標設定はできません。

049 テクスチャードクアッドを回転させる

●回転行列を基にワールド変換を行うことにより回転する

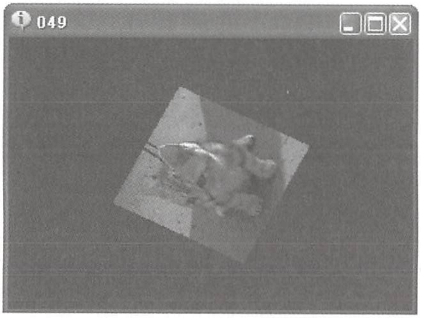
中級

サンプルプロジェクト　049 テクスチャードクアッドを回転

回転行列用の行列を用意して、D3DXMatrixRotationZ関数により回転行列として初期化します。D3DXMatrixRotationZ関数は、指定された角度（ラジアン角）から、Z軸周りの回転行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した回転行列をワールド変換行列に含めて（サンプルではワールド変換は、回転と平行移動の合成変換です）、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

▼実行結果



```
//ワールドトランスフォーム（絶対座標変換）
D3DXMATRIXA16 matWorld,matRotation;
D3DXMatrixTranslation(&matWorld,0,0,1);
D3DXMatrixRotationZ(&matRotation, g_fRoll);
matWorld=matRotation * matWorld;
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
//テクスチャードクアッドのレンダリング
{
    TEX_VERTEX vecPoint[4+1]=
    {
          { -0.5,0.5,0,0,0 }, //頂点1
          {  0.5,0.5,0,1,0 }, //頂点2
          { -0.5,-0.5,0,0,1 },//頂点3
          { 0.5,-0.5,0,1,1 }  //頂点4
```

```
	};

	pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
	pDevice->SetTexture(0,pTexture);
	pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));
}
```

One Point

ここではZ軸回りで回転させましたが、カメラの方向によってはX軸、Y軸で回転させてる場合もあるでしょう。その場合、D3DXMatrixRotationX、あるいはD3DXMatrixRotationY関数により回転行列を作成することになります。

050 テクスチャードクアッドを拡大縮小する

●スケーリング行列を基にワールド変換を行うことによりスケーリングする 初級

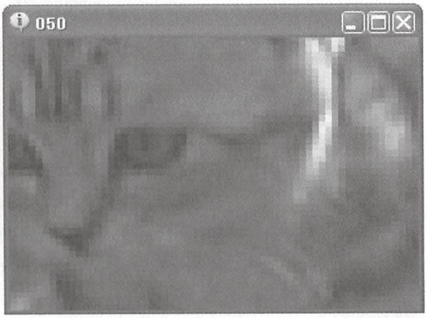
サンプルプロジェクト　050 テクスチャードクアッドを拡大縮小

スケーリング行列用の行列を用意して、D3DXMatrixScaling関数によりスケーリング行列として初期化します。

D3DXMatrixScaling関数は、指定された3つの比率から、スケーリング行列を作成し、第1引数に指定した行列インスタンスに代入します。3つの比率は左から順にX,Y,Z軸方向における縮尺です。

作成したスケーリング行列をワールド変換行列に含めて（サンプルでのワールド変換は、スケーリングと平行移動の合成変換です）、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

▼実行結果



One Point

3つの成分を均等に増減させると均等にスケーリングします。つまり、形の比率はそのままで拡大縮小します。3成分をばらばらに増減させてスケーリングした場合は、縦と横の何れかに歪むのでストレッチと言ったほうが良いかも知れません。

051 テクスチャードクアッドを半透明にする

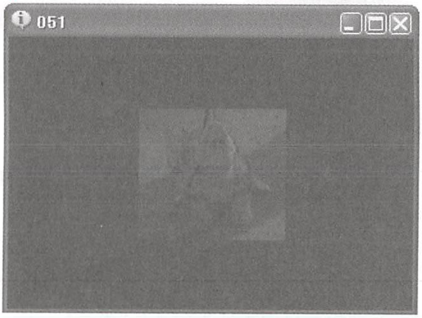
●レンダリグステートの設定と、頂点色のアルファ値の設定で半透明にする

初級

サンプルプロジェクト　051 テクスチャードクアッドを半透明

半透明の仕方はTips041と同様の方法で、レンダリングステートを設定します。

▼実行結果



```
//テクスチャードクアッドのレンダリング
{
    TEX_VERTEX vecPoint[4+1]=
    {
            { -0.5,0.5,0,D3DCOLOR_ARGB(g_bAlpha,255,255,255),0,0 }, //頂点1
            {  0.5,0.5,0,D3DCOLOR_ARGB(g_bAlpha,255,255,255),1,0 }, //頂点2
            { -0.5,-0.5,0,D3DCOLOR_ARGB(g_bAlpha,255,255,255),0,1 },//頂点3
            { 0.5,-0.5,0,D3DCOLOR_ARGB(g_bAlpha,255,255,255),1,1 }  //頂点4
    };

    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_DIFFUSE | D3DFVF_TEX1);
    pDevice->SetTexture(0,pTexture);
    pDevice->SetRenderState (D3DRS_ALPHABLENDENABLE, TRUE);
    pDevice->SetRenderState (D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
    pDevice->SetRenderState (D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);
    pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1,D3DTA_DIFFUSE);
    pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG2,D3DTA_DIFFUSE);

    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));
}
```

One Point

透明にするには、レンダリングステートの変更と共に、頂点にディフューズ成分（アルファ値を含む）を追加します。

052 テクスチャードクアッドの“ふち”を背景色に馴染ませる

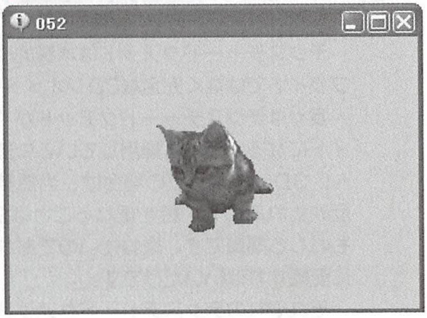
テクスチャのカラーキーとレンダリングステートのアルファテストで色を抜く

中級

サンプルプロジェクト　052 テクスチャードクアッドの“ふち”処理

ここで使用しているビットマップの“ふち”は“青”なので、抜け色を“青”に設定しました。

▼実行結果



```
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite_Blueback.bmp",SPRITE_W
IDTH＊(16.0f /24.0f),SPRITE_HEIGHT＊(16.0f /24.0f),0,0,D3DFMT_UNKNOWN,
D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,
0xff0000ff,NULL,NULL,&pTexture))
```

青が抜けた後、黒くレンダリングされてしまうので、アルファテストにより、“黒”であればそのピクセルを描画しないようにします。

```
//ビデオカードにアルファテスト機能があれば、アルファテスト設定する
D3DCAPS9 Caps;
pDevice->GetDeviceCaps(&Caps);
if (Caps.AlphaCmpCaps & D3DPCMPCAPS_GREATEREQUAL)
{
    //黒(0,0,0) より大きい場合のみラスタライズするように設定
    pDevice->SetRenderState(D3DRS_ALPHAREF, (DWORD)0x00000001);
    pDevice->SetRenderState(D3DRS_ALPHATESTENABLE, TRUE);
    pDevice->SetRenderState(D3DRS_ALPHAFUNC, D3DCMP_GREATEREQUAL);
```

One Point

抜け色は“青”以外のどのような色でも構いません。ただし、綺麗に抜けるのは「単色」である必要があります。

053 テクスチャードクアッドを意図した重ね順で重ねる

●個別のワールドトランスフォームをかけて、3次元的に配置する　中級

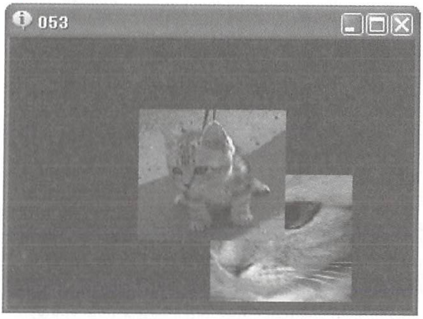
サンプルプロジェクト　053 テクスチャードクアッドを重ねる

テクスチャードクアッドは本質的には、スプライトではなく完全な3Dジオメトリです。

我々はテクスチャードクアッドが“スプライトになるように”運用しているに過ぎません。3Dジオメトリの場合は、当然奥行きは反映されるので、絵を重ねることは2Dよりもむしろ簡単です。重ねたいのであれば、奥に配置すればいいだけです。

それぞれのテクスチャードクアッドに固有のワールドトランスフォームをかけて、奥と手前になるように配置すれば、それだけで重なります。

▼実行結果



```
//テクスチャードクアッドのレンダリング
{
    TEX_VERTEX vecPoint[4+1]=
    {
        { -0.5,0.5,0,0,0 }, //頂点1
        {  0.5,0.5,0,1,0 }, //頂点2
        { -0.5,-0.5,0,0,1 },//頂点3
        { 0.5,-0.5,0,1,1 }  //頂点4
    };

    //1枚目のワールドトランスフォーム（絶対座標変換）
    D3DXMatrixTranslation(&matWorld,0,0,1);
    pDevice->SetTransform(D3DTS_WORLD, &matWorld);
    //1枚目のレンダリング
    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
    pDevice->SetTexture(0,pTexture);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));

    //2枚目のワールドトランスフォーム（絶対座標変換）
    D3DXMatrixTranslation(&matWorld,0.5,-0.5 ,1.1);
    pDevice->SetTransform(D3DTS_WORLD, &matWorld);
    //2枚目のレンダリング
    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
    pDevice->SetTexture(0,pTexture_2);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));
}
```

One Point

3次元物体なので、レンダリング順番を気にすることはなく、どのような順番でレンダリングしようとも、位置的に奥にあるものは常に背面にレンダリングされるので楽です。

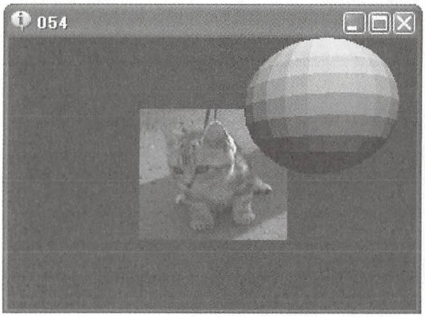
054 メッシュと一緒にレンダリングする

●同時にID3DXMeshインターフェイスでメッシュをレンダリングする

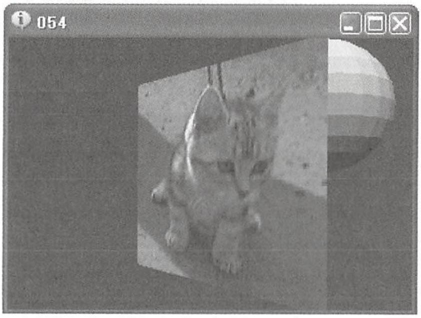
中級

サンプルプロジェクト　054 メッシュと一緒にレンダリング

▼実行結果（54-1）



▼実行結果（54-2）



図ではスフィアメッシュ（球メッシュ）をテクスチャードクアッドと一緒にレンダリングしています。メッシュのレンダリングについては、Tips074以降で解説しています。また、スフィアメッシュの作成についてはTips075、076で解説していますので、そちらを参照してください。

メッシュと一緒にレンダリングするのに特別なことはなにもありません。両方とも3Dジオメトリなので、レンダリングの流れは同一です。

図54-2では、テクスチャードクアッドをカメラに対し傾けています。3Dジオメトリなので姿勢も自由にとることができます。

```
//メッシュのワールドトランスフォーム（絶対座標変換）
D3DXMatrixRotationY(&matRotation, timeGetTime()/3000.0f);
D3DXMatrixTranslation(&matTrans,0.7,0.5,1);
matWorld=matRotation*matTrans;
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
//メッシュのレンダリング
for(DWORD i=0; i<dwNumMaterials; i++)
{
          pDevice->SetMaterial(&pMeshMaterials[i]);
          pDevice->SetTexture(0, pMeshTextures[i]);
    pDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
          pMesh->DrawSubset(i);
}
//テクスチャードクアッドのワールドトランスフォーム（絶対座標変換）
D3DXMATRIXA16 matWorld;
D3DXMatrixTranslation(&matWorld,0,0,1);
//D3DXMatrixRotationY(&matWorld, timeGetTime()/1000.0f);
pDevice->SetTransform(D3DTS_WORLD, &matWorld);
//テクスチャードクアッドのレンダリング
{
    TEX_VERTEX vecPoint[4+1]=
    {
          { -0.5,0.5,0,0,0 }, //頂点1
          {  0.5,0.5,0,1,0 }, //頂点2
          { -0.5,-0.5,0,0,1 },//頂点3
          { 0.5,-0.5,0,1,1 }  //頂点4
    };

    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
    pDevice->SetTexture(0,pTexture);
    pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,vecPoint,sizeof
(TEX_VERTEX));
}
```

なお、常に絵をメッシュの後ろ側にレンダリングしたい場合は、絵のレンダリングの前に

```
pDevice->SetRenderState(D3DRS_ZENABLE,FALSE);
```

とし、絵をメッシュの最初にレンダリングすれば実現できます。絵をレンダリングした後で

```
pDevice->SetRenderState(D3DRS_ZENABLE,FALS);
```

として、Zバッファーを有効にするなどしてください。メッシュについては、Tips073以降のメッシュの部分を参照してください。

055 フルスクリーンに復帰時に再初期化をする

●デバイスをリセットし、かつ、必要なオブジェクトを再度初期化する　中級

サンプルプロジェクト　055 フルスクリーンからの復帰

Alt+tabキーにより、画面をデスクトップに切り替えると、フルスクリーンでのリソースは破壊されます。破壊されたリソース（ここではテクスチャオブジェクト）は元に戻さなくてはなりません。このことの本質については、Tips020を参照してください。

まずは、デバイスをリセットします。

```
//デバイス消失への対応
if(pDevice->Present(NULL, NULL, NULL, NULL)==D3DERR_DEVICELOST)
{
    if(pDevice->TestCooperativeLevel() == D3DERR_DEVICELOST)
    {
        return;
    }
    SAFE_RELEASE(pTexture);
    SAFE_RELEASE(pSprite);

    D3DPRESENT_PARAMETERS d3dpp;

    ZeroMemory(&d3dpp, sizeof(d3dpp));
    D3DDISPLAYMODE DMode;
    pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT ,&DMode);
    d3dpp.BackBufferFormat =DMode.Format;
    d3dpp.BackBufferWidth=WINDOW_WIDTH;
    d3dpp.BackBufferHeight=WINDOW_HEIGHT;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = FALSE;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;

    if(pDevice->Reset(&d3dpp)!=D3D_OK)
    {
        MessageBox(0,"デバイスのリセットに失敗しました","",MB_OK);
    }
```

次にテクスチャを再初期化します。プログラム起動直後の初期化コードと全く同一です。

```
        //「テクスチャオブジェクト」の“再”作成

if(FAILED(D3DXCreateTextureFromFileEx(pDevice,"Sprite.jpg",SPRITE_WIDTH,SPRI
TE_HEIGHT,0,0,
D3DFMT_A8R8G8B8,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,
              0xff000000,NULL,NULL,&pTexture)))
    {
        MessageBox(0,"テクスチャの再構築に失敗しました","",MB_OK);
        return ;
    }
```

次からが、ID3DXSpriteと異なる部分です。

デバイスを失うと、レンダリングステートはリセットされてしまいます。なので、フルスクリーンに復帰した時は、レンダリングステートを再度設定しなくてはなりません。そして、トランスフォーム（ビュー、プロジェクション）も設定し直す必要があります。

本サンプルでは頂点はレンダリング前に毎回構築しているので、特に再構築の必要はありません。

```
// カリングはオフにする（つまり、背面もレンダリングするということ）
pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
// ライトをオフにする
pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
// zバッファー処理を有効にする
pDevice->SetRenderState(D3DRS_ZENABLE, TRUE);
// ビュートランスフォーム（視点座標変換）
D3DXVECTOR3 vecEyePt(0.0f, 0.0f,-1.5f); //カメラ（視点）位置
D3DXVECTOR3 vecLookatPt(0.0f, 0.0f, 0.0f);//注視位置
D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
pDevice->SetTransform(D3DTS_VIEW, &matView);
// プロジェクショントランスフォーム（射影変換）
D3DXMatrixPerspectiveFovLH(&matProj,D3DX_PI/4,(FLOAT)WINDOW_WIDTH/(FLOAT)WIN
DOW_HEIGHT, 1.0f, 1000.0f);
pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
```

One Point

要するに、デバイスをリセットしたあとは、初期化コードをそのままコピーするか、初期化ルーチンを再度呼び出せばいいわけです。

3Dプリミティブ

056 プリミティブとは

●プリミティブの意味と種類

初級

Direct3Dにおけるプリミティブとは、なんらかのジオメトリを形成する最小単位のことであり、

点（ポイント）
線（ライン）
面（三角ポリゴン）

のことを指します。

複数のプリミティブからなるジオメトリをメッシュと呼びます。メッシュについては、Tips073以降のメッシュの部分で解説しています。

ちなみに、3DCG“ソフト”の世界では、基本形形状メッシュをプリミティブと呼びますが、Direct3Dにおいてプリミティブとは、あくまでもメッシュの構成要素であってメッシュではありません。

通常、プリミティブ1つから成るジオメトリは、ゲーム内でほとんど使用しません。単純な四角ポリゴンですらプリミティブ2つから成るメッシュですから、なんらかのジオメトリはメッシュであることがほとんどです。

なお、線は点2つから成り、面（三角ポリゴン）は点3つから成ります。言うなれば、点（ポイント）が究極のプリミティブと言えます。点（ポイント）を線や面の構成要素として捉える時は、頂点（Vertex）とも呼びます。

057 ポイント(点）を作成する

●レンダリングするためにポイント（点）を準備します

初級

参考プロジェクト　058ポイントのレンダリング

ポイント（点）は、最も単純なジオメトリプリミティブです。ポイントはライン（線）とトライアングル（面）の構成要素にもなっています。

もっとも簡単なポイントは、情報として“座標”のみを持つものです。座標のみなので、D3DXVECTOR3すなわち3次元ベクトルただそれだけで、ポイントを用意したことになります。

次のコードは2つのポイントを用意しています。

```
//最も簡単なポイント　D3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[2]=
{
    D3DXVECTOR3(0,0.5,0),
    D3DXVECTOR3(0.5,0.5,0)
```

ポイントは、座標のみならず、いろいろな付帯情報を持つことができます。色、テクスチャ座標、重み...etc。

ここでは、座標と色を持つポイントを定義し、そのインスタンスを2つ作成しています。

```
//定義
struct  COLOR_POINT
{
    D3DXVECTOR3  vecCoord;
    DWORD  dwColor;
};
//色付きのポイント(COLOR_POINT型定義は、コード上部で行っている)
COLOR_POINT  ColorPoint[2]=
{
    D3DXVECTOR3(0,0.3,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.5,0.3,0),D3DCOLOR_ARGB(255,0,255,0)
};
```

One Point

座標のみでも、大抵の場合はポイントの型をアプリケーション内で定義したほうが、統一がとれていいでしょう。

058 ポイント（点）のレンダリング

●IDirect3DDevice9::DrawPrimitiveUP関数により、ポイントをレンダリングする　初級

サンプルプロジェクト　058ポイントのレンダリング

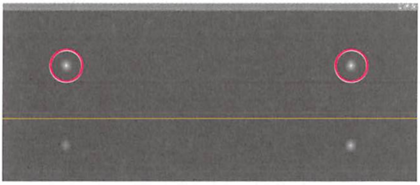
1つのポイントのレンダリング結果は非常に小さいため、画面で確認し辛く、ましてや紙面では確認することが不可能です。そのため、ウィンドウを最大化し、なんとか確認できるようにしました。そのため、ポイントはぼやけて見えますが、拡大していることによる正常な結果です。

まずは、最も簡単に座標のみを用意して、それをレンダリングするコードが次のコード片です。

```
//最も簡単なポイント　D3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[2]=
{
    D3DXVECTOR3(0,0.5,0),
    D3DXVECTOR3(0.5,0.5,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
pDevice->DrawPrimitiveUP(D3DPT_POINTLIST, 2, vecPoint,
sizeof(D3DXVECTOR3));
```

右の図中の上部2つのポイントが、このポイントのレンダリングです。紙面では分かりませんが単色です。

▼実行結果



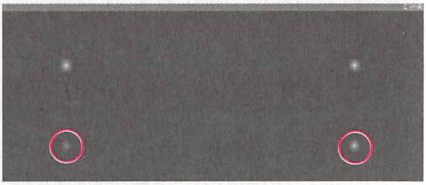
次に、色を持たせたポイントをレンダリングするコードが次のコード片です。

```
//色付きのポイント(COLOR_POINT型定義は、コード上部で行っている)
COLOR_POINT  ColorPoint[2]=
{
    D3DXVECTOR3(0,0.3,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.5,0.3,0),D3DCOLOR_ARGB(255,0,255,0)
};
pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_DIFFUSE );
```

```
pDevice->DrawPrimitiveUP(D3DPT_POINTLIST, 2, ColorPoint,
sizeof(COLOR_POINT));
```

そのレンダリング結果は、右の図中の下部2つのポイントです。紙面では分かりませんが、赤と緑の色が付いています。

▼実行結果



ポイントの準備とレンダリングについて、その流れは全て同じです。

1.なんらかの型を定義する
2.IDirect3DDevice9::SetFVFメソッドにより、型をセットする。
3.IDirect3DDevice9::DrawPrimitiveUPメソッドによりレンダリングする。

というものです。

IDirect3DDevice9::SetFVFに渡しているフラグに注目してください。座標のみの場合と色つきの場合では、異なっています。座標のみの場合は最も単純で座標を意味するD3DFVF_XYZフラグのみを渡します。そして、色をつけたい場合は、D3DFVF_XYZフラグと併せてD3DFVF_DIFFUSEフラグも渡します。ディフューズとは、色成分の拡散反射のことであり、結果的に表面色を意味します。

ポイントとしてレンダリングする場合、DrawPrimitiveUPメソッドの第1引数には、D3DPT_POINTLISTフラグを渡します。

One Point

ポイントのみのレンダリングは、モデリングツール等の制作でもない限り、あまり行わないような気がします。

059 ライン（線）を作成する

レンダリングするためにライン（線）を準備する

初級

参考プロジェクト　060 ラインのレンダリング

ラインの作成（データの準備）と言っても、ライン独自のデータ型があるわけではなく、ポイントを用意するだなので、ポイントの準備と全く同じです。

点としてレンダリングするか、点を結んだ線としてレンダリグするかは、レンダリング段階で指定します。

最も単純なラインの準備

```
//最も簡単なライン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[2]=
{
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(0.5,0.4,0)
};
```

色付きラインの準備

```
//定義
struct  COLOR_POINT
{
    D3DXVECTOR3  vecCoord;
    DWORD  dwColor;
};

//色付きのライン　色付き頂点を使用
COLOR_POINT  ColorPoint[2]=
{
    D3DXVECTOR3(-0.5,0.3,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.5,0.2,0),D3DCOLOR_ARGB(255,0,255,0)
};
```

One Point

ラインはポイントを結んだものであるので、用意するものはポイントです。

060 ライン（線）のレンダリング

IDirect3DDevice9::DrawPrimitiveUP関数を使う

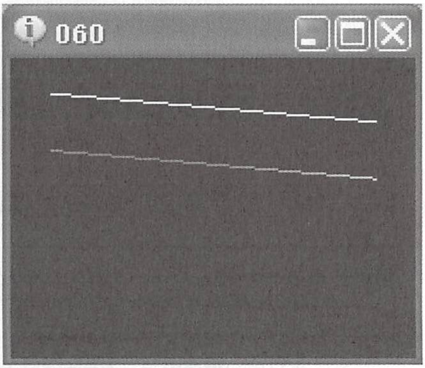
初級

サンプルプロジェクト　060　ラインのレンダリング

紙面では分かりませんが、上のラインは色なし、下のラインが色情報を持つラインです。正確に表現すると、上のラインは色なしポイントを結んだもの、下のラインが色情報を持つポイントを結んだもの、と言えます。

上のラインのレンダリングコードは次のとおりです。

▼実行結果



```
//最も簡単なライン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[2]=
{
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(0.5,0.4,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
pDevice->DrawPrimitiveUP(D3DPT_LINELIST, 1, vecPoint, sizeof(D3DXVECTOR3));
```

下の色つきラインのレンダリングコードは次のとおりです。

```
//色付きのライン　色付き頂点を使用
COLOR_POINT  ColorPoint[2]=
{
    D3DXVECTOR3(-0.5,0.3,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.5,0.2,0),D3DCOLOR_ARGB(255,0,255,0)
};
pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_DIFFUSE  );
pDevice->DrawPrimitiveUP(D3DPT_LINELIST,1,                    ColorPoint,
sizeof(COLOR_POINT));
```

データ自体（ポイント）は同じものを、DrawPrimitiveUPメソッドにD3DPT_LINELISTフラグを渡すことによりラインとしてレンダリングする、言い換えると、それらのポイントを結ぶようにレンダリングすることを指示しています。

One Point

ラインは、レイの確認用としてレンダリングすることがよくあります。

061 ポリゴン（面）を作成する

●レンダリングするためにポリゴン（面）を準備する

中級

参考プロジェクト　062 ポリゴンのレンダリング

ポリゴンの作成（データの準備）と言っても、ポリゴン独自のデータ型があるわけではなく、ポイントを用意するだなので、ポイントの準備と全く同じです。

点としてレンダリングするか、点を結んで中身を塗りつぶすポリゴンとしてレンダリグするかは、レンダリング段階で指定します。

●最も単純なポリゴンの準備

```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
```

●色付きポリゴンの準備

```
struct  COLOR_POINT
{
    D3DXVECTOR3  vecCoord;
    DWORD  dwColor;
};

//色付きのポリゴン　色付き頂点を使用
```

```
COLOR_POINT  ColorPoint[3]=
{
    D3DXVECTOR3(0,0.2,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.25,0.6,0),D3DCOLOR_ARGB(255,0,255,0),
    D3DXVECTOR3(0.5,0.2,0),D3DCOLOR_ARGB(255,0,0,255)
};
```

●テクスチャを貼ることのできるポリゴンの準備

```
struct  TEX_POINT
{
    D3DXVECTOR3  vecCoord;
    D3DXVECTOR2  vecUV;
};

//テクスチャー付きのポリゴン　テクスチャー座標付き頂点を使用
TEX_POINT  TexPoint[4]=
{
    D3DXVECTOR3(0,0,0),D3DXVECTOR2(0,0),
    D3DXVECTOR3(0.5,0,0),D3DXVECTOR2(1,0),
    D3DXVECTOR3(0,-0.5,0),D3DXVECTOR2(0,1),
    D3DXVECTOR3(0.5,-0.5,0),D3DXVECTOR2(1,1),
};
```

One Point

ポリゴンはポイントで囲まれた領域を塗りつぶすものであるので、用意するものはポイントです。

062 ポリゴン（面）のレンダリング

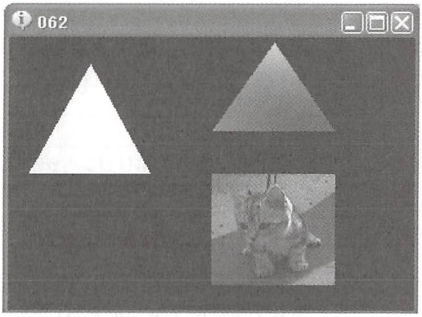
●IDirect3DDevice9::DrawPrimitiveUP関数を使う

中級

サンプルプロジェクト　062 ポリゴンのレンダリング

データ自体（ポイント）は同じものを、DrawPrimitiveUPメソッドにD3DPT_TRIANGLELISTフラグを渡すことによりポリゴンとしてレンダリングする、言い換えると、それらのポイントで囲んだ領域を塗りつぶすようにレンダリングすることを指示しています。

▼実行結果



●左側の三角形

```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
     D3DXVECTOR3(-0.75,0,0),
     D3DXVECTOR3(-0.5,0.5,0),
     D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,        1,          vecPoint,
sizeof(D3DXVECTOR3));
```

ポリゴンは頂点3つの3角形が最小単位です。DrawPrimitiveUPメソッドの第2引数はプリミティブの数を意味します。この場合は1つの三角形をレンダリングしたいので1とします。

SetFVFメソッドにはD3DFVF_XYZフラグにより座標のみを設定します。

●右上の三角形

```
//色付きのポリゴン　色付き頂点を使用
COLOR_POINT  ColorPoint[3]=
{
    D3DXVECTOR3(0,0.2,0),D3DCOLOR_ARGB(255,255,0,0),
    D3DXVECTOR3(0.25,0.6,0),D3DCOLOR_ARGB(255,0,255,0),
    D3DXVECTOR3(0.5,0.2,0),D3DCOLOR_ARGB(255,0,0,255)
};
pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_DIFFUSE  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,1,              ColorPoint,
sizeof(COLOR_POINT));
```

これも1つの三角形だけをレンダリングします。

SetFVFメソッドにはD3DFVF_XYZフラグとD3DFVF_DIFFUSEフラグにより座標と色を設定します。

●右下の四角形

```
//テクスチャー付きのポリゴン　テクスチャー座標付き頂点を使用
TEX_POINT  TexPoint[4]=
{
    D3DXVECTOR3(0,0,0),D3DXVECTOR2(0,0),
    D3DXVECTOR3(0.5,0,0),D3DXVECTOR2(1,0),
    D3DXVECTOR3(0,-0.5,0),D3DXVECTOR2(0,1),
    D3DXVECTOR3(0.5,-0.5,0),D3DXVECTOR2(1,1),
};
pDevice->SetTexture(0,pTexture);
pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_TEX1  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,                  TexPoint,
sizeof(TEX_POINT));
```

これは、2つの三角形により4角形を表現しています。三角形2つなので第2引数は2とします。

三角形を2つレンダリングするのに、頂点を4つしか用意していないことに留意してください。

これは、D3DPT_TRIANGLESTRIPフラグによりトライアングル・ストリップという規則により三角形をレンダリングしているからです。

SetFVFメソッドには、D3DFVF_XYZフラグとD3DFVF_TEX1フラグにより座標とテクスチャ座標を設定します。

OnePoint

トライアングルのレンダリング規則はトライアングル・リスト、トライアングル・ストリップ、トライアングル・ファンがあります。トライアングル・リストは三角形の数×3だけ頂点が必要になります。トライアングル・ファンは最も少ない頂点数で複数の三角形をレンダリングできます。

063 プリミティブのマテリアル（色、光沢等）を設定する

●D3DMATERIAL9マテリアル構造体を初期化し、設定する

中級

サンプルプロジェクト　063 プリミティブのマテリアル

現実世界において物体が目に見えるのは、光（ライト）が反射するからであり（物体に反射した光が目に入るからであり）、光がなければ何も見ることができません（自己発光するものは除く）。Direct3Dにおいて、ライティングをオフにしても（光が無くても）レンダリング結果が見えるのは、ある意味、現実に反したCG特有のものであることを認識することが大切です。CG特有のライトとは具体的には「アンビエント光」によりレンダリングです。これはDirect3Dに限らず3DCG全般に当てはまります。

さてマテリアルは、ジオメトリの表面上での光の反射具合のことです。当然、ライティング計算を行ってはじめて反映されます（効果を確認できます）。ライティング計算は、レンダリングステート（ID3Device9::SetRenderState）でD3DRS_LIGHTINGをTRUEにすることにより有効になります。また、ライティング計算には法線情報が必要であり、法線により、ディフューズ反射（色と陰影）とスペキュラー反射（光沢、ハイライト）の計算が可能になります。

したがって、ジオメトリが法線を持っていないとライティング計算が行えませんので、頂点を定義する場合は法線情報を持つように定義し、そして、SetFVFにより型を通知する時は、D3DFVF_NORMALフラグを指定してやる必要があります。

マテリアルにおけるディフューズ計算によりジオメトリ表面の“陰影及び色“が決定されることになり、また、スペキュラー計算により”光沢（ハイライト）“言い換えれば”てかり具合“が決定されます。

平たく表現すると、マテリアルは”物質表面の材質感”のパラメーターです。

サンプルではディフューズ（拡散）反射で、赤と青の光のみを拡散反射するようにしています。その結果、表面の色は紫になります（紙面では分かりませんが）。

```
Materials.Diffuse.r=1.0f;
Materials.Diffuse.g=0.0f;
Materials.Diffuse.b=1.0f;
Materials.Diffuse.a=1.0f;
```

そして、スペキュラー反射は、三原色全てを均等に反射するので光沢（ハイライト）は白色になります。

```
Materials.Specular.r=1.0f;
Materials.Specular.g=1.0f;
Materials.Specular.b=1.0f;
```

また、ハイライトの大きさは中くらいに設定しました。

```
Materials.Power=50;
```

ライトを最低1個作成し、光がポリゴン表面に当たるように方向あるいは位置、またはその両方を適切に設定しなくてはなりません。

```
D3DLIGHT9  light;
ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
(省略)
pDevice->SetLight(  0,  &light  );
pDevice->LightEnable(  0,  TRUE  );
```

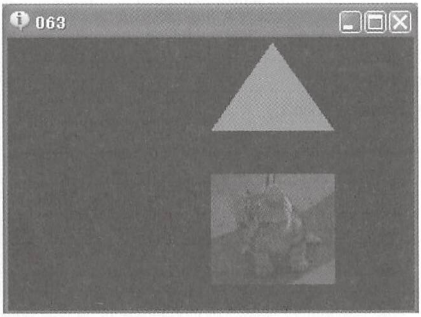
先述のとおりライティング計算をするように指示しなくてはなりません。

```
//  ライトを有効にしてマテリアルを計算（反映）させるようにする
pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
```

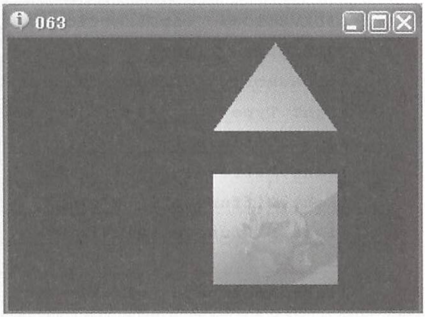
また、スペキュラーも反映されますが、反映させるかどうかは任意です。反映させると表面に光沢が出ます。

```
//スペキュラー（鏡面反射）
pDevice->SetRenderState(  D3DRS_SPECULARENABLE,  TRUE  );
```

▼スペキュラーを無効にした場合



▼スペキュラーを有効にした場合



```
//定義
struct  COLOR_POINT
{
     D3DXVECTOR3  vecCoord;
     D3DXVECTOR3  vecNormal;
};
```

```
struct  TEX_POINT
{
    D3DXVECTOR3  vecCoord;
    D3DXVECTOR3  vecNormal;
    D3DXVECTOR2  vecTex;
};
```

```
//  ライトを有効にしてマテリアルを計算（反映）させるようにする
pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
//スペキュラー（鏡面反射）
pDevice->SetRenderState(  D3DRS_SPECULARENABLE,  TRUE  );
```

```
//マテリアル
D3DMATERIAL9  Materials;
ZeroMemory(&Materials,sizeof(Materials));
Materials.Diffuse.r=1.0f;
Materials.Diffuse.g=0.0f;
Materials.Diffuse.b=1.0f;
Materials.Diffuse.a=1.0f;
Materials.Specular.r=1.0f;
Materials.Specular.g=1.0f;
Materials.Specular.b=1.0f;
Materials.Power=50;
pDevice->SetMaterial(&Materials);
//  ライトをあてる　白色で鏡面反射ありに設定
D3DXVECTOR3  vecDirection(0,0,1);
D3DLIGHT9  light;
ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
light.Type             =  D3DLIGHT_DIRECTIONAL;
light.Diffuse.r    =  1.0f;
light.Diffuse.g    =  1.0f;
light.Diffuse.b    =  1.0f;
light.Diffuse.a  =1.0f;

light.Ambient.a  =1.0f;

light.Specular.r=1.0f;
light.Specular.g=1.0f;
light.Specular.b=1.0f;

D3DXVec3Normalize(  (D3DXVECTOR3＊)&light.Direction,  &vecDirection  );
light.Range  =  200.0f;
pDevice->SetLight(  0,  &light  );
```

```
pDevice->LightEnable(  0,  TRUE  );

pDevice->Clear(         0,         NULL,          D3DCLEAR_TARGET         |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,0),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
{

    //法線付きのポリゴン  法線付き頂点を使用
    COLOR_POINT  ColorPoint[3]=
    {
        D3DXVECTOR3(0,0.2,0),D3DXVECTOR3(0,0,-1),//頂点1
        D3DXVECTOR3(0.25,0.6,0),D3DXVECTOR3(0,0,-1),//頂点2
        D3DXVECTOR3(0.5,0.2,0),D3DXVECTOR3(0,0,-1)//頂点3
    };

    pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_NORMAL  );
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,1,              ColorPoint,
sizeof(COLOR_POINT));

    //テクスチャー付きのポリゴン  テクスチャー座標及び法線付き頂点を使用
    TEX_POINT  TexPoint[4]=
    {
        D3DXVECTOR3(0,0,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(0,0),
        D3DXVECTOR3(0.5,0,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(1,0),
        D3DXVECTOR3(0,-0.5,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(0,1),
        D3DXVECTOR3(0.5,-0.5,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(1,1),
    };
    pDevice->SetTexture(0,pTexture);
    pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_NORMAL  |  D3DFVF_TEX1  );
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,              TexPoint,
sizeof(TEX_POINT));
    pDevice->SetTexture(0,NULL);

    pDevice->EndScene();
}
```

One Point

メッシュのマテリアルも、プリミティブのマテリアルと全く同じ考え方および設定になります。そもそもメッシュはプリミティブの集まりなので当然ですね。

064 プリミティブを半透明でレンダリングする

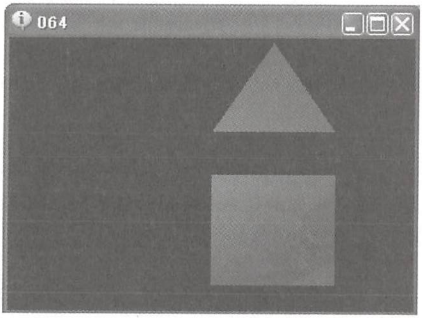
マテリアルのディフューズのアルファ成分とレンダリングステートを適切に設定する　中級

サンプルプロジェクト　064 プリミティブを半透明

半透明にする場合は、マテリアルのディフューズのアルファ成分を1未満にすると同時に、レンダリングステートもアルファブレンディングが効くように設定しなくてはなりません。

マテリアル、ディフューズのアルファ成分を1未満に。サンプルでは0.5に設定。

▼実行結果



```
Materials.Diffuse.a=0.5f;
```

レンダリングステートもアルファブレンディングモードに設定。

```
//ポリゴンを半透明にするレンダリングステート
pDevice->SetRenderState  ( D3DRS_ALPHABLENDENABLE,  TRUE  );
pDevice->SetRenderState  ( D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA  );
pDevice->SetRenderState  ( D3DRS_DESTBLEND,  D3DBLEND_INVSRCALPHA  );
```

サンプルではテクスチャを貼ったポリゴンもレンダリングしていますが、テクスチャも半透明にする場合は次のステートも指定する必要があります。

```
//テクスチャを貼ったポリゴンも半透明にしたい場合のレンダリングステート
pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1,D3DTA_DIFFUSE);
pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG2,D3DTA_DIFFUSE);
```

One Point

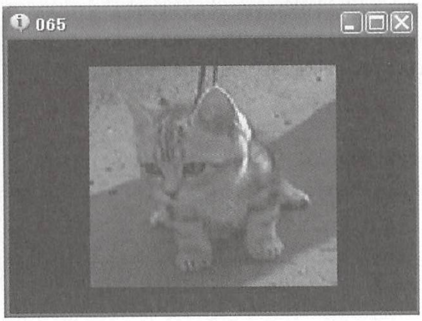
表面の透明度を決定するのはディフューズのアルファ成分（a成分）だけです。サンプルではa=0.5fなので50%半透明でレンダリングされます

065 テクスチャを貼ったプリミティブをレンダリングする

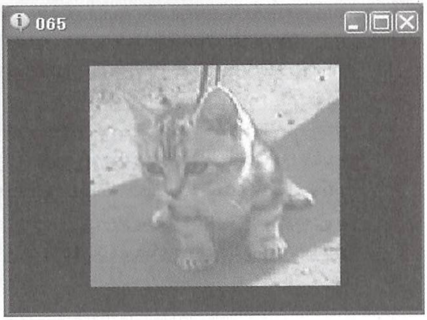
●ID3Device9::SetTextureによりジオメトリにテクスチャを適用する 中級

サンプルプロジェクト 065 プリミティブにテクスチャを貼る

▼スペキュラー無効



▼スペキュラー有効



テクスチャを貼り付けるための手順は次のとおりです。

1．テクスチャ座標を持つ頂点を定義する
2．テクスチャオブジェクトを初期化する
3．ID3Device9::SetTextureメソッドを実行する

1．テクスチャ座標を持つ頂点を定義する

```
//定義
struct  TEX_POINT
{
    D3DXVECTOR3  vecCoord;
    D3DXVECTOR3  vecNormal;
    D3DXVECTOR2  vecTex;
};
```

2．テクスチャオブジェクトを初期化する

```
//「テクスチャオブジェクト」の作成
f(FAILED(D3DXCreateTextureFromFileEx(pDevice,”Sprite.jpg”,SPRITE_WIDTH＊
(16.0f/24.0f),SPRITE_HEIGHT＊(16.0f/24.0f),0,0,D3DFMT_UNKNOWN,D3DPOOL_DEFAULT
,D3DX_FILTER_NONE,D3DX_DEFAULT,0xff000000,NULL,NULL,&pTexture)))
{
    MessageBox(0,”テクスチャの作成に失敗しました”,””,MB_OK);
    return  E_FAIL;
}
```

3. ID3Device9::SetTextureメソッドを実行する

```
//テクスチャー付きのポリゴン　テクスチャー座標及び法線付き頂点を使用
TEX_POINT  TexPoint[4]=
{
    D3DXVECTOR3(-1,1,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(0,0),
    D3DXVECTOR3(1,1,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(1,0),
    D3DXVECTOR3(-1,-1,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(0,1),
    D3DXVECTOR3(1,-1,0),D3DXVECTOR3(0,0,-1),D3DXVECTOR2(1,1),
};
pDevice->SetTexture(0,pTexture);
pDevice->SetFVF(D3DFVF_XYZ  |  D3DFVF_NORMAL  |  D3DFVF_TEX1  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,                    TexPoint,
sizeof(TEX_POINT));
pDevice->SetTexture(0,NULL);
```

One Point

SetTextureは、それ以降のジオメトリに適用されます。したがって、別のSetTextureにより別のテクスチャがセットされるまで、レンダリングされるあらゆるジオメトリには最初のテクスチャが貼り付けられます。

066 プリミティブを移動する

●移動行列を基にワールド変換を行うことにより平行移動する　中級

サンプルプロジェクト　066　プリミティブを移動する

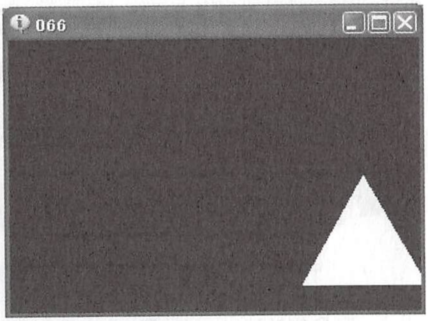
平行移動用の行列を用意して、D3DXMatrixTranslation関数により平行移動行列として初期化します。

D3DXMatrixTranslation関数は、指定されたX,Y,Z3つの座標成分から、平行移動行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した平行移動行列をワールド変換行列に含めて（サンプルではほかの変換がないので、そのままワールド行列とも言えます）、そのワールド行列をIDirect3DDevice9::SetTransform(D3DTS_WORLD, OOO)メ

ソッドによりワールド変換として登録します。

▼実行結果



```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//ワールドトランスフォーム
D3DXMATRIX  matWorld;
D3DXMatrixTranslation(&matWorld,g_vecPosition.x,g_vecPosition.y,g_vecPosition.z);
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST, 1, vecPoint, sizeof(D3DXVECTOR3));
```

One Point

D3DXMatrixTranslationの「Translation」は、平行移動という意味です。

067 プリミティブを回転させる

回転行列を基にワールド変換を行うことにより回転する

中級

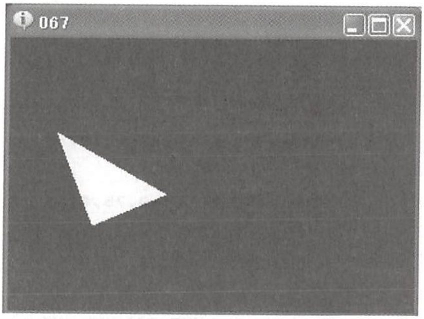
サンプルプロジェクト　067　プリミティブを回転させる

回転行列用の行列を用意して、D3DXMatrixRotationX関数、D3DXMatrixRotationY関数、D3DXMatrixRotationZ関数により回転行列として初期化します。

3つの関数は、指定された角度（ラジアン角）から、それぞれX軸、Y軸、Z周りの回転行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した回転行列をワールド変換行列に含めて、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

▼実行結果



```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//ワールドトランスフォーム（絶対座標変換）
D3DXMATRIX  matWorld,matYaw,matPitch,matRoll;
D3DXMatrixRotationX(  &matPitch,  g_vecRotation.x  );
D3DXMatrixRotationY(  &matYaw,  g_vecRotation.y  );
D3DXMatrixRotationZ(  &matRoll,  g_vecRotation.z  );
matWorld=matYaw  *  matPitch  *  matRoll;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,  1,  vecPoint,
sizeof(D3DXVECTOR3));
```

One Point

2Dスプライトの回転軸は1つだけでしたが、3D物体の回転軸は無数にあります。

068 プリミティブを拡大縮小する

スケーリング行列を基にワールド変換を行う

中級

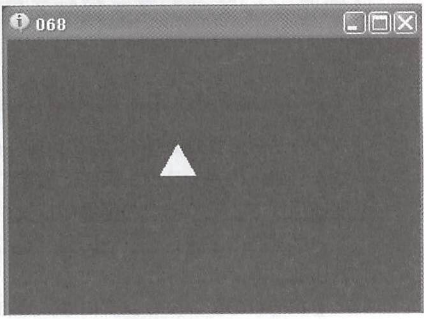
サンプルプロジェクト 068 プリミティブを拡大縮小する

スケーリング行列用の行列を用意して、D3DXMatrixScaling関数によりスケーリング行列として初期化します。

D3DXMatrixScaling関数は、指定された3つの比率から、スケーリング行列を作成し、第1引数に指定した行列インスタンスに代入します。3つの比率は左から順にX,Y,Z軸方向における縮尺です。

作成したスケーリング行列をワールド変換行列に含めて、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

▼実行結果



```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//ワールドトランスフォーム
D3DXMATRIX  matWorld,matScale;
D3DXMatrixScaling(&matScale,g_vecScale.x,g_vecScale.y,g_vecScale.z);
matWorld=matScale;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
```

```
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST, 1, vecPoint,
sizeof(D3DXVECTOR3));
```

One Point

3つの成分を均等に増減させると均等にスケーリングします。つまり、形の比率はそのままで拡大縮小します。3成分をばらばらに増減させてスケーリングした場合は、縦と横の何れかに歪むのでストレッチと言ったほうが良いかも知れません。

069　1つのプリミティブを複数レンダリングする

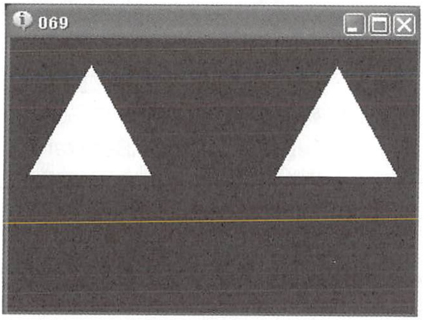
1つのインスタンスを異なる座標にレンダリングして複数に見せる　中級

サンプルプロジェクト　069　1つのプリミティブを複数レンダリング

プリミティブの実態そのものは単一のもの、つまり単一のインスタンスに、複数の座標を関連付けて、それぞれの座標にレンダリングすれば複数のオブジェクトに見せることができます。

それぞれのレンダリングには、それぞれのワールド変換を行います。

▼実行結果



```
//最も簡単なポリゴン　頂点にD3DXVECTOR3型だけを、そのまま使用
D3DXVECTOR3  vecPoint[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//最初のワールドトランスフォーム
```

```
D3DXMATRIX  matWorld;
D3DXMatrixTranslation(&matWorld,g_vecPosition.x,g_vecPosition.y,g_vecPositio
n.z);
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,  1,  vecPoint,
sizeof(D3DXVECTOR3));
//別のワールドトランスフォームをかけて、別の位置にレンダリング
D3DXMatrixTranslation(&matWorld,g_vecPosition.x+1,g_vecPosition.y,g_vecPosit
ion.z);
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,  1,  vecPoint,
sizeof(D3DXVECTOR3));
```

One Point

同じ"絵"であるキャラクター等は、このように"絵"のインスタンスが1つである場合もあります。

ただし、アニメーションメッシュにおいては、全てが同じモーションになってしまうので、インスタンス自体を固有に持たせなければなりません。

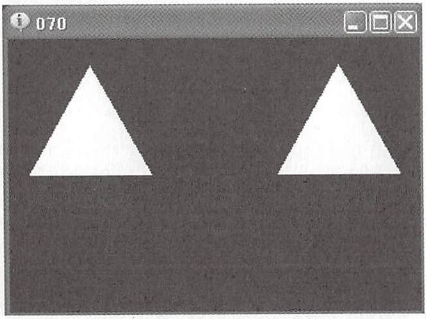
070 複数のプリミティブをレンダリングする

●プリミティブのインスタンス自体を個々に用意してレンダリングする 中級

サンプルプロジェクト 070 複数のプリミティブをレンダリング

Tips69と同様です。ただ、プリミティブが2つあるので頂点は2×3=6個用意する部分が異なるだけです。

▼実行結果



```
//1個目の頂点初期化
D3DXVECTOR3  vecPoint[3]=

{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//ワールドトランスフォーム
D3DXMatrixTranslation(&matWorld,0,0,0);
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
//レンダリング
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,        1,        vecPoint,
sizeof(D3DXVECTOR3));

//2個目の頂点初期化
D3DXVECTOR3  vecPoint2[3]=
{
    D3DXVECTOR3(-0.75,0,0),
    D3DXVECTOR3(-0.5,0.5,0),
    D3DXVECTOR3(-0.25,0,0)
};
pDevice->SetFVF(D3DFVF_XYZ  );
//ワールドトランスフォーム
D3DXMatrixTranslation(&matWorld,1,0,0);
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
//レンダリング
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,        1,        vecPoint2,
sizeof(D3DXVECTOR3));
```

One Point

アニメーションメッシュや、形状の異なる“絵”をもつキャラクターをレンダリングする場合は、当然それぞれについて、インスタンスを持たせなければなりません。同じ絵の使い回しはできないということです。

071 フルスクリーンに復帰した時に再初期化をする

●デバイスをリセットし、同時に必要なオブジェクトを再度初期化する 中級

サンプルプロジェクト 071 フルスクリーンからの復帰

Alt+tabキーにより、画面をデスクトップに切り替えると、フルスクリーンでのリソースは破壊されます。破壊されたリソース（ここではテクスチャオブジェクト）は元に戻さなくてはなりません。このことの本質については、Tips020を参照してください。

デバイスを失うと、レンダリングステートはリセットされてしまいます。なので、フルスクリーンに復帰した時は、レンダリングステートを再度設定しなくてはなりません。そして、トランスフォーム（ビュー、プロジェクション）も設定し直す必要があります。

本サンプルでは頂点はレンダリング前に毎回構築しているので、特に再構築の必要はありません。トランスフォームも特に重要ではないので、コードには書いていません。

```
pDevice->Clear(         0,         NULL,          D3DCLEAR_TARGET       |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,0),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
{
    //最も簡単なポリゴン  頂点にD3DXVECTOR3型だけを、そのまま使用
    D3DXVECTOR3  vecPoint[3]=
    {
          D3DXVECTOR3(-0.75,0,0),
          D3DXVECTOR3(-0.5,0.5,0),
          D3DXVECTOR3(-0.25,0,0)
    };
    pDevice->SetFVF(D3DFVF_XYZ  );
    //ワールドトランスフォーム
    D3DXMATRIX  matWorld;

D3DXMatrixTranslation(&matWorld,g_vecPosition.x,g_vecPosition.y,g_vecPositio
n.z);
    pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,         1,        vecPoint,
sizeof(D3DXVECTOR3));

    pDevice->EndScene();
}
//デバイス消失への対応
```

```
if(pDevice->Present(  NULL,  NULL,  NULL,  NULL  )==D3DERR_DEVICELOST)
{
    if(pDevice->TestCooperativeLevel()  ==  D3DERR_DEVICELOST)
    {
        return;
    }

    D3DPRESENT_PARAMETERS  d3dpp;

    ZeroMemory(  &d3dpp,  sizeof(d3dpp)  );
    D3DDISPLAYMODE  DMode;
    pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT  ,&DMode);
    d3dpp.BackBufferFormat  =DMode.Format;
    d3dpp.BackBufferWidth=WINDOW_WIDTH;
    d3dpp.BackBufferHeight=WINDOW_HEIGHT;
    d3dpp.BackBufferCount=1;
    d3dpp.SwapEffect  =  D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed  =  FALSE;
    d3dpp.EnableAutoDepthStencil  =  TRUE;
    d3dpp.AutoDepthStencilFormat  =  D3DFMT_D16;

    if(pDevice->Reset(&d3dpp)!=D3D_OK)
    {
        MessageBox(0,"デバイスのリセットに失敗しました","",MB_OK);
    }
    //  カリングはオフにする（つまり、背面もレンダリングするということ）
    pDevice->SetRenderState(  D3DRS_CULLMODE,  D3DCULL_NONE  );
    //  ライトをオフにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,  FALSE  );
}
```

One Point

要するに、デバイスをリセットしたあとは、初期化コードをそのままコピーするか、初期化ルーチンを再度呼び出せばいいわけです。

072 プリミティブをビルボーディングする

ビュー行列の逆行列によりビルボーディングをかける

初級

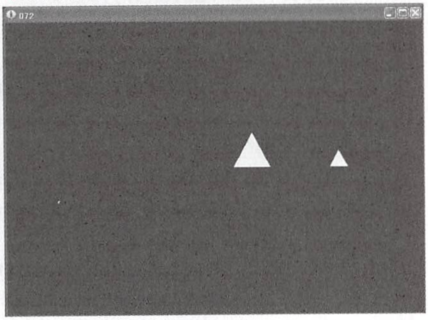
サンプルプロジェクト 072 プリミティブをビルボーディング

右図において、左側のプリミティブがビルボーディングをかけたプリミティブ、すなわちビルボードです。

本サンプルは矢印キーでカメラ（視点）が回転するようになっています。右側のプリミティブはビルボーディングをかけていない通常のものなので、当然回転します。それに対し、左のビルボードプリミティブは、視点がどうであろうと常に同じ姿勢に留まります。

ビルボーディングは、その時点のビュー変換の逆行列をかけることにより実現します。それにより、それ以降のレンダリングは、カメラの影響を受けません。

▼実行結果



```
VOID  Render()
{
     D3DXMATRIX  matWorld;
     D3DXMATRIX  matView;
     D3DXMATRIX  matProj;
     //ワールドトランスフォーム（絶対座標変換）
     D3DXMatrixTranslation(  &matWorld,1,0,5  );
     pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
     //  ビュートランスフォーム（視点座標変換）
     D3DXVECTOR3  vecEyePt=vecCamera;  //カメラ（視点）位置
     D3DXVECTOR3  vecLookatPt=vecCamera+D3DXVECTOR3(  0.0f,  0.0f,  1.0f  );//
注視位置
     D3DXVECTOR3  vecUpVec(  0.0f,  1.0f,  0.0f  );//上方位置
     D3DXMatrixLookAtLH(  &matView,  &vecEyePt,  &vecLookatPt,  &vecUpVec  );
     pDevice->SetTransform(  D3DTS_VIEW,  &matView  );
     //  プロジェクショントランスフォーム（射影変換）
     D3DXMatrixPerspectiveFovLH(          &matProj,                 D3DX_PI/4,
(FLOAT)WINDOW_WIDTH/(FLOAT)WINDOW_HEIGHT,  0.1f,  1000.0f  );
     pDevice->SetTransform(  D3DTS_PROJECTION,  &matProj  );
```

```
        pDevice->Clear(      0,      NULL,       D3DCLEAR_TARGET   |
    D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,100),  1.0f,  0  );
        if(  SUCCEEDED(  pDevice->BeginScene()  )  )
        {
                //1個目のプリミティブのレンダリング（通常）
                pDevice->SetFVF(D3DFVF_XYZ  );
                pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,    1,    g_vecPoint,
    sizeof(D3DXVECTOR3));

                //2個目のプリミティブのレンダリング（ビルボーディングをかける）
                D3DXMATRIX  matBillboard;
                D3DXMatrixInverse(&matBillboard,NULL,&matView);//ビルボーディング用打
    ち消し行列作成
                matWorld*=matBillboard;
                pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

                pDevice->SetFVF(D3DFVF_XYZ  );
                pDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST,    1,    g_vecPoint,
    sizeof(D3DXVECTOR3));


                pDevice->EndScene();
        }
        pDevice->Present(  NULL,  NULL,  NULL,  NULL  );
    }
```

One Point

実際的に言うと、カメラの影響を受けないのではなく、カメラの影響を受けてちょうど元に戻るような変換がなされます。結果的に影響を受けていないように見えるということです。

073 メッシュとは

●メッシュとは複数のプリミティブから構成されるジオメトリ

初級

メッシュとは、複数のプリミティブから構成されるジオメトリです。複数のプリミティブを繋ぎ合わせて、何らかの形状を形作ったもので、様々な物体やキャラクターのビジュアルとして利用します。

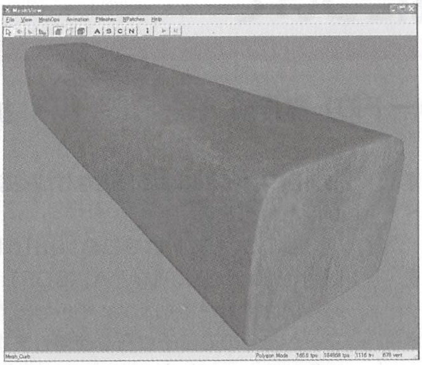
例えば、左の図は道路の縁石を表現するメッシュです。

メッシュを構成しているものはプリミティブです。

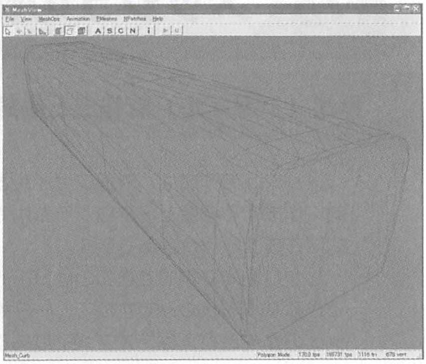
左下の図は、1つのプリミティブを選択した状態です。

この縁石メッシュは、3DCGソフトでモデリングしたものをXファイルに変換したものです。

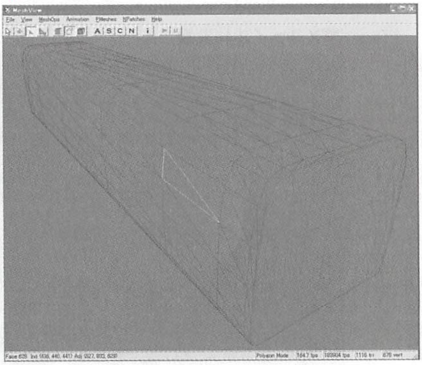
▼縁石メッシュ


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N
Mesh_Curb
Polygon Mode

▼プリミティブの集まり


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N
Mesh_Curb
Polygon Mode

▼プリミティブ（トライアングル）


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N
Polygon Mode

この縁石メッシュなどのように、3DCGソフトとモデリングが必要なメッシュの他に、球やボックスなどの基本形状メッシュは、算術的に生成できるので、関数コールだけで生成したりします。

算術的に自動生成できるメッシュは、球（スフィア）、ボックス、トーラス、シリンダー、ティーポット、及びテキストフォントがあり、それらの生成についてはTips075から086で解説しています。

Xファイルメッシュについては、Tips087、088で解説しています。メッシュはプリミティブの集まりですから、プリミティブのようにマテリアルにより材質感を作成したり、テクスチャを貼り付けることも当然可能です。実際この縁石メッシュは、コンクリートの画像をテクスチャとして貼っています。

One Point

たいてい3Dゲームでは、メッシュを使用します。

074 メッシュの作成について

静的（ファイル）な作成と動的（コード的）な作成

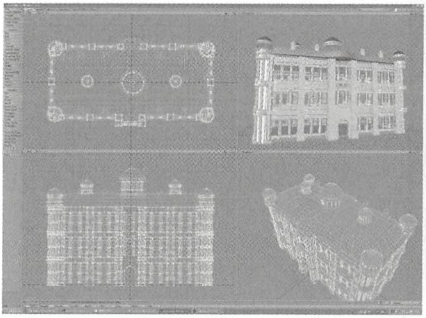
初級

メッシュの作成は、外部ファイルとして静的に作成する場合と、プログラム中でコード的に作成する場合があります。

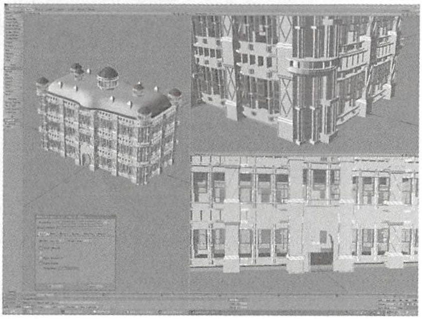
しかし動的に作成できるのは算術的に生成できる基本形状のみですので、一般的なメッシュの作成と言えば、Xファイルあるいはオリジナルフォーマットファイル、何れにしても外部ファイルによる静的な作成を指すことがほとんどです。

3DCGソフトは3DStudioMAX、LightWave3D、MAYA、SOFTIMAGEあるいはフリーでもXファイルを出力するソフトがあるので、それらを使用します。つまり最低限、モデリング及びXファイルを出力

▼モデリング



▼Xファイルに変換



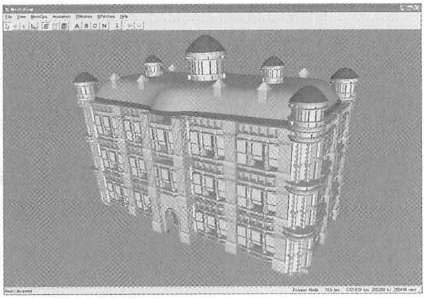
できるツールを用意しなくてはならないということです。

筆者はLightWave3Dを9年間使用しています。前ページの図はLightWaveでモデリングし、Xファイルに出力する過程です。

Xファイルとして出力したら、Direct3Dで読み込むことができるメッシュとなります。右の図は、上で作成したメッシュを、Direct3Dのメッシュビューアーで出力したものです。

一方、このようなツールにより外部ファイルとして用意しなくても“基本形状”のメッシュであれば、関数により自動生成できます。自動生成できる形状とその生成関数は次のとおりです。それぞれについては、Tips075から086を参照してください。

▼実行結果



立方体（ボックス）→D3DXCreateBox関数
円柱（シリンダー）→D3DXCreateCylinder関数
球（スフィア）→D3DXCreateSphere関数
茶瓶（ティーポット）→D3DXCreateTeapot関数
3Dテキストフォント→D3DXCreateText関数
ドーナツ形状（トーラス）→D3DXCreateTorus関数

One Point

3DCGソフトは高価なものですが、本当に3Dを研究したいのであれば、持っていないとかえって高くつきますよ（自分自身の労力＝人件費）。

075 スフィア（球）を作成する

●D3DXCreateSphere関数を使う

初級

参考プロジェクト　076　スフィアのレンダリング

綺麗に形の整った基本形状は、算術計算により生成できるので、整っていればいるほど、むしろ簡単に作成できます。

基本形状であるスフィアは、その作成関数であるD3DXCreateSphere関数によりコード上で動的に作成することができます。

例えば次のように作成したりします。

```
LPD3DXMESH  pSphereMesh=NULL;
    D3DXCreateSphere(pDevice,1,24,24,&pSphereMesh,NULL);
```

引数の意味は次のとおりです。

第1引数　デバイスオブジェクトのポインター

第2引数　球の半径です。0.0f 以上である必要があります。

第3引数　スライス数。スライスとは縦の分割を意味します。スライス数は“横”線となって反映されます。

第4引数　スタック数。スタックとは横の分割を意味します。スタック数は“縦”線となって反映されます。

第5引数　D3DXMESHインターフェイスポインタのアドレス。

第6引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。

＊＊＊＊＊＊

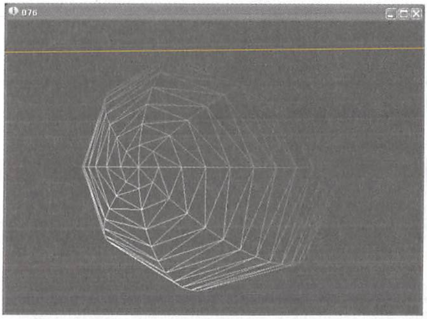
スライスを8に変更すると左下の図のようになります。

```
D3DXCreateSphere(pDevice,1,8,24,&pSphereMesh,NULL);
```

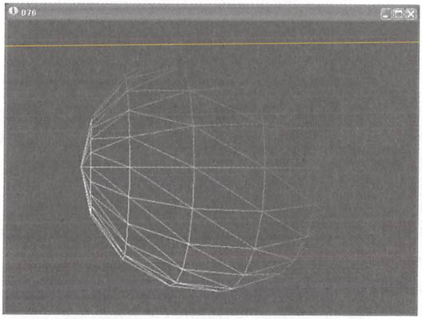
今度はスライスを24に戻し、スタックを8に変更すると右下の図のようになります。

```
D3DXCreateSphere(pDevice,1,24,8,&pSphereMesh,NULL);
```

▼スライス（縦の分割）＝8（横線）
スタック（横の分割）＝24（縦線）



▼スライス（縦の分割）＝24（横線）
スタック（横の分割）＝8（縦線）



One Point

スライスとスタックをあまり大きくするとポリゴン数（プリミティブ数）が増え、余計な時間がかかります。見栄えに違いがでない程度の低い数で十分でしょう。

076 スフィア（球）のレンダリング

●スフィアメッシュのDrawSubsetメソッドを使う

初級

サンプルプロジェクト 076 スフィアのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
LPD3DXMESH  pSphereMesh=NULL;
```

メッシュポインターを用意します。

```
//スフィアの作成
    D3DXCreateSphere(pDevice,1,24,24,&pSphereMesh,NULL);
```

作成はD3DXCreateSphere関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pSphereMesh->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応”レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

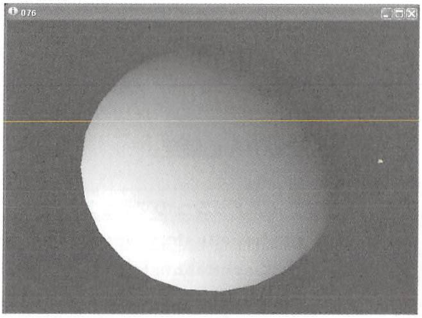
●マテリアルの設定

```
//スフィアメッシュのマテリアル
    pSphereMeshMaterials  =  new  D3DMATERIAL9;
    pSphereMeshMaterials->Diffuse.r=1.0f;
    pSphereMeshMaterials->Diffuse.g=1.0f;
    pSphereMeshMaterials->Diffuse.b=1.0f;
    pSphereMeshMaterials->Diffuse.a=1.0f;
    pSphereMeshMaterials->Ambient=pSphereMeshMaterials->Diffuse;
    pSphereMeshMaterials->Specular.r=1.0f;
    pSphereMeshMaterials->Specular.g=1.0f;
    pSphereMeshMaterials->Specular.b=1.0f;
    pSphereMeshMaterials->Emissive.r=0.1f;
    pSphereMeshMaterials->Emissive.g=0.1f;
    pSphereMeshMaterials->Emissive.b=0.1f;
    pSphereMeshMaterials->Power=120.0f;
```

●ライトの設定

```
//  ライトをオンにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
    //  ライトをあてる  白色で光沢ありに設定
      D3DXVECTOR3  vecDirection(1,1,1);
          D3DLIGHT9  light;
          ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
          light.Type                 =  D3DLIGHT_DIRECTIONAL;
          light.Diffuse.r      =  1.0f;
          light.Diffuse.g      =  1.0f;
          light.Diffuse.b      =  1.0f;
      light.Specular.r=1.0f;
      light.Specular.g=1.0f;
      light.Specular.b=1.0f;
         D3DXVec3Normalize(  (D3DXVECTOR3*)&light.Direction,  &vecDirection
);
          light.Range                =  200.0f;
          pDevice->SetLight(  0,  &light  );
          pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



One Point

球のように曲面から成る形状の場合、ランバートやフォンモデルのシェーディングを行ったほうが（つまりライティングを有効）見栄えがいいです。

077 ボックス（立方体）を作成する

●D3DXCreateBox関数でボックスメッシュを生成する

初級

参考プロジェクト　078　ボックスのレンダリング

綺麗に形の整った基本形状は、算術計算により逆に簡単に作成できます。

基本形状であるボックスは、その作成関数であるD3DXCreateBox関数によりコード上で動的に作成することができます。ボックスの生成は基本形状の中でも最も簡単です。

例えば次のように作成したりします。

```
LPD3DXMESH  pBoxMesh=NULL;
    D3DXCreateBox(pDevice,1,1,1,&pBoxMesh,NULL);
```

引数の意味は次のとおりです。

第1引数　デバイスオブジェクトのポインター

第2引数　x軸方向のボックスの幅。

第3引数　y軸方向のボックスの高さ。

第4引数　z軸方向のボックスの深さ。

第5引数　D3DXMESHインターフェイスポインタのアドレス。

第6引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。

078 ボックス（立方体）のレンダリング

●DrawSubsetメソッドを使う

初級

サンプルプロジェクト　078　ボックスのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
LPD3DXMESH  pBoxMesh=NULL;
```

メッシュポインターを用意します。

```
//ボックスの作成
    D3DXCreateBox(pDevice,1,1,1,&pBoxMesh,NULL);
```

作成はD3DXCreateBox関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pBoxMesh->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応“レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

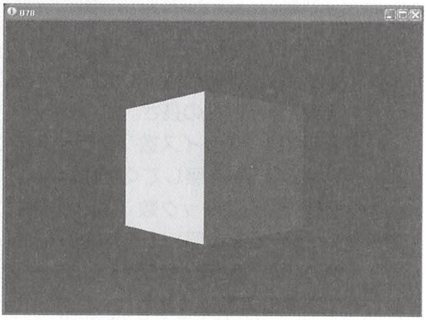
●マテリアルの設定

```
//ボックスメッシュのマテリアル
    pBoxMeshMaterials  =  new  D3DMATERIAL9;
    pBoxMeshMaterials->Diffuse.r=1.0f;
    pBoxMeshMaterials->Diffuse.g=1.0f;
    pBoxMeshMaterials->Diffuse.b=1.0f;
    pBoxMeshMaterials->Diffuse.a=1.0f;
    pBoxMeshMaterials->Ambient=pBoxMeshMaterials->Diffuse;
    pBoxMeshMaterials->Specular.r=1.0f;
    pBoxMeshMaterials->Specular.g=1.0f;
    pBoxMeshMaterials->Specular.b=1.0f;
    pBoxMeshMaterials->Emissive.r=0.1f;
    pBoxMeshMaterials->Emissive.g=0.1f;
    pBoxMeshMaterials->Emissive.b=0.1f;
    pBoxMeshMaterials->Power=120.0f;
```

●ライトの設定

```
//  ライトをオンにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
    //  ライトをあてる  白色で光沢ありに設定
      D3DXVECTOR3  vecDirection(1,1,1);
          D3DLIGHT9  light;
          ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
          light.Type              =  D3DLIGHT_DIRECTIONAL;
          light.Diffuse.r    =  1.0f;
          light.Diffuse.g    =  1.0f;
          light.Diffuse.b    =  1.0f;
      light.Specular.r=1.0f;
      light.Specular.g=1.0f;
      light.Specular.b=1.0f;
          D3DXVec3Normalize(  (D3DXVECTOR3 *)&light.Direction,  &vecDirection
    );
            light.Range             =  200.0f;
            pDevice->SetLight(  0,  &light  );
            pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



One Point

曲面のないボックスのような形状の場合においてライトがディレクショナルの場合、フラットシェードなのかグーローシェード（あるいはフォン）なのか分かりづらいですが、ライトをポイントまたはスポットにすると違いがわかります。

079 シリンダー（円柱）を作成する

●D3DXCreateCylinder関数を使う

初級

参考プロジェクト　080 シリンダーのレンダリング

綺麗に形の整った基本形状は、算術計算により逆に簡単に作成できます。

基本形状であるシリンダーは、その作成関数であるD3DXCreateCylinder関数によりコード上で動的に作成することができます。

例えば次のように作成したりします。

```
LPD3DXMESH   pCylinderMesh=NULL;
    D3DXCreateCylinder(pDevice,1,1,3,24,2,&pCylinderMesh,NULL);
```

引数の意味は次のとおりです。シリンダーはZ軸に平行となるように作成されます。

第1引数　デバイスオブジェクトのポインター。

第2引数　手前の円（断面）の半径。

第3引数　奥側の円（断面）の半径。

第4引数　円柱の長さ。

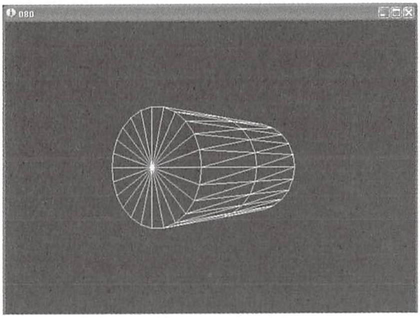
第5引数　スライス数（スライスについてはTips075を参照してください）。

第6引数　スタック数（スタックについてはTips075を参照してください）。

第7引数　D3DXMESHインターフェイスポインタのアドレス。

第8引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。

▼円柱



＊＊＊＊＊＊

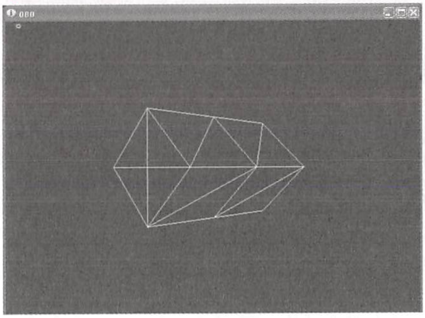
スライスを4に変更すると、下の左の図のように“四角柱”になります。

```
D3DXCreateCylinder(pDevice,1,1,3,4,2,&pCylinderMesh,NULL);
```

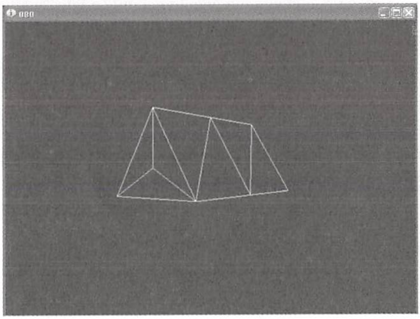
スライスを3に変更すると、下の右の図のように“三角柱”になります。

```
D3DXCreateCylinder(pDevice,1,1,3,3,2,&pCylinderMesh,NULL);
```

▼四角柱



▼三角柱



One Point

シリンダーでは、スタック数はあまり意味がありません。1でも十分なくらいです。

080 シリンダー（円柱）のレンダリング

●スフィアメッシュのDrawSubsetメソッドを使う

初級

サンプルプロジェクト　080　シリンダーのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
LPD3DXMESH  pBoxMesh=NULL;
```

メッシュポインターを用意します。

```
//シリンダーの作成
    D3DXCreateCylinder(pDevice,1,1,3,24,2,&pCylinderMesh,NULL);//円柱
```

作成はD3DXCreateCylinder関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pCylinderMesh->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応“レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

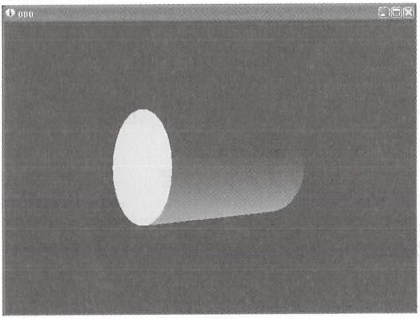
●マテリアルの設定

```
//シリンダーメッシュのマテリアル
    pCylinderMeshMaterials  =  new  D3DMATERIAL9;
    pCylinderMeshMaterials->Diffuse.r=1.0f;
    pCylinderMeshMaterials->Diffuse.g=1.0f;
    pCylinderMeshMaterials->Diffuse.b=1.0f;
    pCylinderMeshMaterials->Diffuse.a=1.0f;
    pCylinderMeshMaterials->Ambient=pCylinderMeshMaterials->Diffuse;
    pCylinderMeshMaterials->Specular.r=1.0f;
    pCylinderMeshMaterials->Specular.g=1.0f;
    pCylinderMeshMaterials->Specular.b=1.0f;
    pCylinderMeshMaterials->Emissive.r=0.1f;
    pCylinderMeshMaterials->Emissive.g=0.1f;
    pCylinderMeshMaterials->Emissive.b=0.1f;
    pCylinderMeshMaterials->Power=120.0f;
```

●ライトの設定

```
//  ライトをオンにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
    //  ライトをあてる  白色で光沢ありに設定
      D3DXVECTOR3  vecDirection(1,1,1);
          D3DLIGHT9  light;
          ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
          light.Type              =  D3DLIGHT_DIRECTIONAL;
          light.Diffuse.r    =  1.0f;
          light.Diffuse.g    =  1.0f;
          light.Diffuse.b    =  1.0f;
      light.Specular.r=1.0f;
      light.Specular.g=1.0f;
      light.Specular.b=1.0f;
          D3DXVec3Normalize(  (D3DXVECTOR3＊)&light.Direction,  &vecDirection
);
          light.Range              =  200.0f;
          pDevice->SetLight(  0,  &light  );
          pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



081 トーラス（ドーナツ形）を作成する

●D3DXCreateSphere関数でトーラスメッシュを生成する 初級

参考プロジェクト 082 トーラスのレンダリング

綺麗に形の整った基本形状は、算術計算により逆に簡単に作成できます。

基本形状であるトーラスは、その作成関数であるD3DXCreateTorus関数によりコード上で動的に作成することができます。

例えば次のように作成したりします。

```
LPD3DXMESH   pTorusMesh=NULL;
    D3DXCreateTorus(pDevice,0.5,1,12,24,&pTorusMesh,NULL);
```

厚みを外周半径（全体の半径）の2分の1にしています。引数の意味は次のとおりです。

第1引数　デバイスオブジェクトのポインター

第2引数　トーラス内部（穴）の半径。値は 0.0f 以上である必要があります。

第3引数　トーラス外部の半径。値は 0.0f 以上である必要があります。

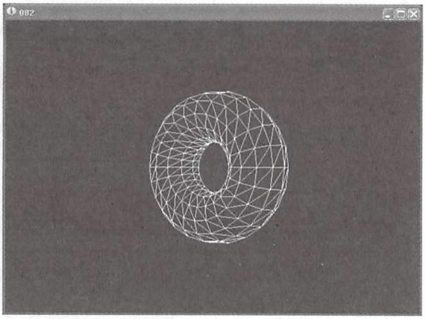
第4引数　横断面の辺の数。値は 3 以上である必要があります。

第5引数　トーラスを構成する環の数。値は 3 以上である必要があります。

第6引数　D3DXMESHインターフェイスポインタのアドレス。

第7引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。

▼実行結果



＊＊＊＊＊＊

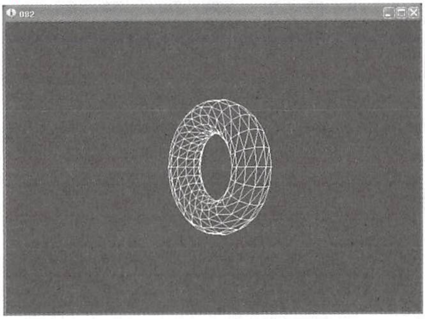
厚みを外周半径（全体の半径）の3分の1にすると次ページの右ののようになります。

```
D3DXCreateTorus(pDevice,0.33,1,12,24,&pTorusMesh,NULL);
```

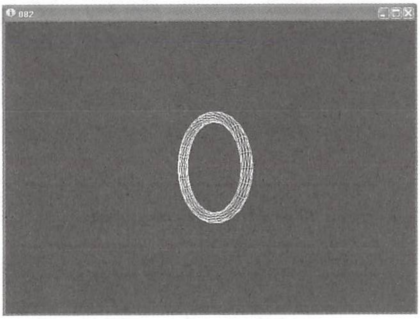
厚みを外周半径（全体の半径）の10分の1にすると次ページのひだ入りの図のようになります。

```
D3DXCreateTorus(pDevice,0.1,1,12,24,&pTorusMesh,NULL);
```

▼厚みが外周半径の3分の1



▼厚みが外周半径の10分の1



One Point

サイドとリングをあまり大きくするとポリゴン数（プリミティブ数）が増え、余計な時間がかかります。見栄えに違いがでない程度の低い数で十分でしょう。

082 トーラス（ドーナツ形）のレンダリング

●DrawSubsetメソッドを使う

初級

サンプルプロジェクト　082 トーラスのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
pTorusMesh   ->DrawSubset(0);
```

メッシュポインターを用意します。

```
//トーラスの作成
    D3DXCreateTorus(pDevice,0.1,1,12,24,&pTorusMesh,NULL);
```

作成はD3DXCreateTorus関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pTorusMesh   ->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応“レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

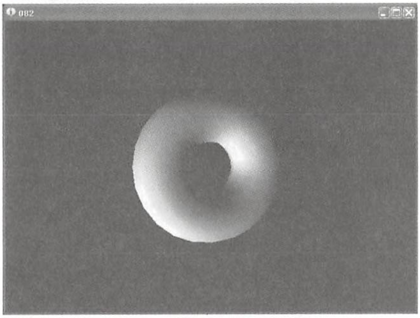
●マテリアル設定

```
//トーラスメッシュのマテリアル
    pTorusMeshMaterials   =  new  D3DMATERIAL9;
    pTorusMeshMaterials->Diffuse.r=1.0f;
    pTorusMeshMaterials->Diffuse.g=1.0f;
    pTorusMeshMaterials->Diffuse.b=1.0f;
    pTorusMeshMaterials->Diffuse.a=1.0f;
    pTorusMeshMaterials->Ambient=pTorusMeshMaterials->Diffuse;
    pTorusMeshMaterials->Specular.r=1.0f;
    pTorusMeshMaterials->Specular.g=1.0f;
    pTorusMeshMaterials->Specular.b=1.0f;
    pTorusMeshMaterials->Emissive.r=0.1f;
    pTorusMeshMaterials->Emissive.g=0.1f;
    pTorusMeshMaterials->Emissive.b=0.1f;
    pTorusMeshMaterials->Power=120.0f;
```

●ライトの設定

```
//  ライトをオンにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,   TRUE   );
    //  ライトをあてる  白色で光沢ありに設定
      D3DXVECTOR3  vecDirection(1,1,1);
          D3DLIGHT9  light;
          ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
          light.Type              =  D3DLIGHT_DIRECTIONAL;
          light.Diffuse.r    =  1.0f;
          light.Diffuse.g    =  1.0f;
          light.Diffuse.b    =  1.0f;
      light.Specular.r=1.0f;
      light.Specular.g=1.0f;
      light.Specular.b=1.0f;
         D3DXVec3Normalize(  (D3DXVECTOR3＊)&light.Direction,  &vecDirection
);
          light.Range             =  200.0f;
          pDevice->SetLight(  0,  &light  );
          pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



One Point

トーラスのように曲面から成る形状の場合、ランバートやフォンモデルのシェーディングを行ったほうが（つまりライティングを有効）見栄えがいいです。

083 ティーポットを作成する

●D3DXCreateTeapot関数でティーポットメッシュを生成する 初級

参考プロジェクト　084 ティーポットのレンダリング

ティーポットは、その作成関数であるD3DXCreateTeapot関数によりコード上で動的に作成することができます。

例えば次のように作成したりします。

```
LPD3DXMESH  pTeapotMesh=NULL;
    D3DXCreateTeapot(pDevice,&pTeapotMesh,NULL);
```

引数の意味は次のとおりです。

第1引数　デバイスオブジェクトのポインター

第2引数　D3DXMESHインターフェイスポインタのアドレス。

第3引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。

One Point

ティーポットは他の基本形状のように算術的に生成されるのではなく、Xファイルメッシュのように頂点等のデータを内部に持っているものです。そのため、生成されるメッシュは固定的であり、細かなパラメーターを必要としません（指定できません）。

084 ティーポットのレンダリング

DrawSubsetメソッドを使う

初級

サンプルプロジェクト　084 ティーポットのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
LPD3DXMESH  pBoxMesh=NULL;
```

メッシュポインターを用意します。

```
//ティーポットの作成
    D3DXCreateTeapot(pDevice,&pTeapotMesh,NULL);
```

作成はD3DXCreateTeapot関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pTeapotMesh->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応“レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

●マテリアルの設定

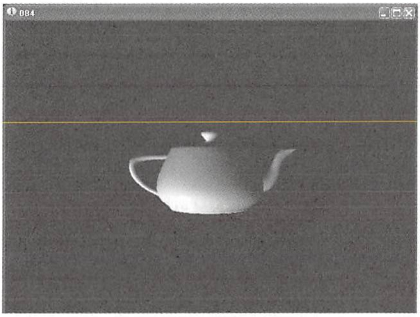
```
//ティーポットメッシュのマテリアル
    pTeapotMeshMaterials  =  new  D3DMATERIAL9;
    pTeapotMeshMaterials->Diffuse.r=1.0f;
    pTeapotMeshMaterials->Diffuse.g=1.0f;
    pTeapotMeshMaterials->Diffuse.b=1.0f;
    pTeapotMeshMaterials->Diffuse.a=1.0f;
    pTeapotMeshMaterials->Ambient=pTeapotMeshMaterials->Diffuse;
    pTeapotMeshMaterials->Specular.r=1.0f;
    pTeapotMeshMaterials->Specular.g=1.0f;
    pTeapotMeshMaterials->Specular.b=1.0f;
    pTeapotMeshMaterials->Emissive.r=0.1f;
    pTeapotMeshMaterials->Emissive.g=0.1f;
```

```
      pTeapotMeshMaterials->Emissive.b=0.1f;
      pTeapotMeshMaterials->Power=120.0f;
```

●ライトの設定

```
  //  ライトをオンにする
      pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
      //  ライトをあてる　白色で光沢ありに設定
        D3DXVECTOR3  vecDirection(1,1,1);
            D3DLIGHT9  light;
            ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
            light.Type              =  D3DLIGHT_DIRECTIONAL;
            light.Diffuse.r     =  1.0f;
            light.Diffuse.g     =  1.0f;
            light.Diffuse.b     =  1.0f;
        light.Specular.r=1.0f;
        light.Specular.g=1.0f;
        light.Specular.b=1.0f;
            D3DXVec3Normalize(  (D3DXVECTOR3＊)&light.Direction,  &vecDirection
  );
            light.Range                =  200.0f;
            pDevice->SetLight(  0,  &light  );
            pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



One Point

ティーポットは、曲面を持ち、かつ、ある程度複雑のある形状なのでシェーディングや映り込みのレンダリングテストには便利でしょう。

085 文字ジオメトリを作成する

D3DXCreateText関数でテキストフォントメッシュを生成する 初級

参考プロジェクト 086 文字ジオメトリのレンダリング

3Dテキストフォントは、その作成関数であるD3DXCreateText関数によりコード上で動的に作成することができます。

3Dテキストを作成するためには、CreateFontIndirect関数等により事前にフォントを作成する必要があります。フォントはトゥルータイプフォントでなければなりません。

```
HDC  hdc  =  CreateCompatibleDC(  NULL  );
HFONT  hFont;
LOGFONT  lf;
ZeroMemory(&lf,  sizeof(lf));
lf.lfHeight  =  16;
lf.lfStrikeOut  =  1;
lstrcpy(lf.lfFaceName,  “tahoma” );
hFont  =  CreateFontIndirect(&lf);
SelectObject(hdc,  hFont);
```

フォントを作成したらD3DXCreateText関数により3Dテキストを生成します。

```
LPD3DXMESH  pTextMesh=NULL;

D3DXCreateText(pDevice,hdc,”     This     is     test”     ,0.1f,
0.2f,&pTextMesh,NULL,NULL);
```

3Dテキストを作成し終えたら、もはやフォントは不要なのでデリートします。

```
DeleteObject(  hFont  );
DeleteDC(  hdc  );
```

D3DXCreateText関数の引数の意味は次のとおりです。

第1引数　デバイスオブジェクトのポインター。

第2引数　フォントのコンテキストハンドル。

第3引数　文字列のポインタ。

第4引数　TrueType フォント アウトラインからの最大偏差。大きい値にする（例えば10.0）とアウトラインからずれてもいい許容範囲が増加し、でき上がるメッシュのポイ

ント・ポリゴン数を低く抑えることができます。ただし、その分シェイプが粗くなります。反対に低い値にすると（例えば0.001）アウトラインからずれる許容範囲が狭くなり、ポイント・ポリゴン数が多く生成されることになりますが、その分アウトラインに正確で滑らかなメッシュとなります。

第5引数　テキストの厚み。
第6引数　D3DXMESHインターフェイスポインタのアドレス。
第7引数　属性バッファへのポインタ。属性バッファを特に用意していない場合はNULL。
第8引数　グリフサイズ。

One Point

グリフとは、文字の形です。普段なにげなく文字のことを“フォント”と言いますが、実は正確ではなく、厳密に言えば、文字の形そのものは“グリフ”であり、フォントは一定のデザインであるグリフの集合を意味します。

086 文字ジオメトリのレンダリング

●DrawSubsetメソッドを使う

初級

サンプルプロジェクト　086 文字ジオメトリのレンダリング

直前のTipsのように、まずメッシュをコード上で作成します。

```
LPD3DXMESH  pBoxMesh=NULL;
```

メッシュポインターを用意します。

```
//テキストの作成
HDC  hdc  =  CreateCompatibleDC(  NULL  );
HFONT  hFont;
LOGFONT  lf;
ZeroMemory(&lf,  sizeof(lf));
lf.lfHeight  =  16;
lf.lfStrikeOut  =  1;
lstrcpy(lf.lfFaceName,  “tahoma”);
hFont  =  CreateFontIndirect(&lf);
SelectObject(hdc,  hFont);

D3DXCreateText(pDevice,hdc,”This is test”,0.001f,
0.2f,&pTextMesh,NULL,NULL);

DeleteObject(  hFont  );
DeleteDC(  hdc  );
```

作成はD3DXCreateText関数を実行すれば一発でできます。

なお、pDeviceは適正に初期化されているデバイスオブジェクトのポインタとします。

```
pTextMesh->DrawSubset(0);
```

レンダリングの本質はここまでです。この2行により“一応“レンダリングはできます。

ただ、サンプルおよび、通常のレンダリングではディフューズ（拡散反射）での滑らかな陰影や、スペキュラー（鏡面反射）による光沢を計算することがほとんどですので、マテリアルとライトを設定します。

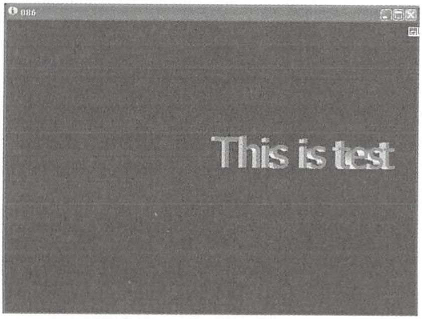
●マテリアルの設定

```
//テキストメッシュのマテリアル
    pTextMeshMaterials  =  new  D3DMATERIAL9;
    pTextMeshMaterials->Diffuse.r=1.0f;
    pTextMeshMaterials->Diffuse.g=1.0f;
    pTextMeshMaterials->Diffuse.b=1.0f;
    pTextMeshMaterials->Diffuse.a=1.0f;
    pTextMeshMaterials->Ambient=pTextMeshMaterials->Diffuse;
    pTextMeshMaterials->Specular.r=1.0f;
    pTextMeshMaterials->Specular.g=1.0f;
    pTextMeshMaterials->Specular.b=1.0f;
    pTextMeshMaterials->Emissive.r=0.1f;
    pTextMeshMaterials->Emissive.g=0.1f;
    pTextMeshMaterials->Emissive.b=0.1f;
    pTextMeshMaterials->Power=120.0f;
```

●ライトの設定

```
//  ライトをオンにする
    pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
    pDevice->SetRenderState(  D3DRS_FILLMODE,D3DFILL_WIREFRAME);
    //pDevice->SetRenderState(  D3DRS_CULLMODE  ,D3DCULL_NONE);
    pDevice->SetRenderState(D3DRS_AMBIENT,0xffffffff);
    //  ライトをあてる  白色で光沢ありに設定
      D3DXVECTOR3  vecDirection(1,1,1);
          D3DLIGHT9  light;
          ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
          light.Type               =  D3DLIGHT_DIRECTIONAL;
          light.Diffuse.r     =  1.0f;
          light.Diffuse.g     =  1.0f;
          light.Diffuse.b     =  1.0f;
      light.Specular.r=1.0f;
      light.Specular.g=1.0f;
      light.Specular.b=1.0f;
         D3DXVec3Normalize(  (D3DXVECTOR3＊)&light.Direction,  &vecDirection
);
          light.Range               =  200.0f;
          pDevice->SetLight(  0,  &light  );
          pDevice->LightEnable(  0,  TRUE  );
```

▼実行結果



One Point

3Dテキストは、生成時のコードが他の基本形状よりも多少増えます。

087 メッシュをXファイルから読み込む

●D3DXLoadMeshFromX関数を使う

中級

参考プロジェクト　088　メッシュをレンダリングする

メッシュのインスタンス、及びマテリアルとテクスチャのインスタンスを宣言しておきます。

```
LPD3DXMESH  pMesh  =  NULL;
D3DMATERIAL9＊  pMeshMaterials  =  NULL;
LPDIRECT3DTEXTURE9＊  pMeshTextures   =  NULL;

DWORD  dwNumMaterials  =  0;
```

あと、マテリアル数を保持する変数も用意しておきます。

```
//  xファイルからメッシュをロードする
LPD3DXBUFFER  pD3DXMtrlBuffer  =  NULL;

if(  FAILED(  D3DXLoadMeshFromX(  “Chips.x”,  D3DXMESH_SYSTEMMEM,
                        pDevice,  NULL,  &pD3DXMtrlBuffer,  NULL,
          &dwNumMaterials,  &pMesh  )  )  )
{
          MessageBox(NULL,  “xファイルの読み込みに失敗しました”,NULL,  MB_OK);
            return  E_FAIL;
}
```

D3DXLoadMeshFromX関数は、メッシュのジオメトリ的な生成は行ってくれますが、質感を表すマテリアルとテクスチャについてはアプリケーション側で次のように生成してやらなければなりません。

```
D3DXMATERIAL＊    d3dxMaterials    =    (D3DXMATERIAL＊  )pD3DXMtrlBuffer-
>GetBufferPointer();
pMeshMaterials  =  new  D3DMATERIAL9[dwNumMaterials];
pMeshTextures   =  new  LPDIRECT3DTEXTURE9[dwNumMaterials];

for(  DWORD  i=0;  i<dwNumMaterials;  i++  )
{
          pMeshMaterials[i]  =  d3dxMaterials[i].MatD3D;
                pMeshMaterials[i].Ambient  =  pMeshMaterials[i].Diffuse;
                pMeshTextures[i]  =  NULL;
                if(  d3dxMaterials[i].pTextureFilename  !=  NULL  &&
                        lstrlen(d3dxMaterials[i].pTextureFilename)  >  0  )
                {
                          if(  FAILED(  D3DXCreateTextureFromFile(  pDevice,
d3dxMaterials[i].pTextureFilename,                    &pMeshTextures[i]  )  )  )
                        {
                              MessageBox(NULL,  “テクスチャの読み込みに失敗しまし
た”,  NULL,  MB_OK);
               }
                }
        }
```

```
pD3DXMtrlBuffer->Release();
```

なお、テクスチャを使用しないメッシュの場合は、テクスチャ関連の初期化部分は不要です。ただ、汎用性を考えて、テクスチャ部分を書いておいて損はないでしょう。

One Point

D3DXLoadMeshFromX関数を実行した後もアプリケーション側のコードがけっこう必要ですが、これはある程度決まりきったコードです、このコードは他のプロジェクトでもそのまま使用できると思います。

088 Xファイルメッシュをレンダリングする

DrawSubsetメソッドを使う

中級

サンプルプロジェクト　088 メッシュをレンダリングする

Xファイルからロードしたメッシュのレンダリングのコードは次のようになります。

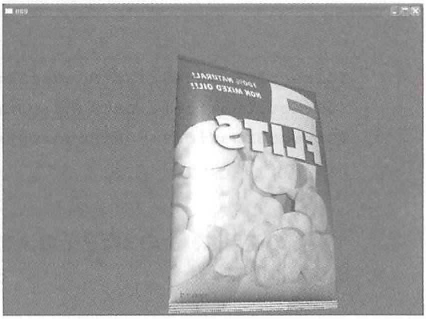
```
    //ワールドトランスフォーム（絶対座標変換）
     D3DXMATRIX  matWorld,matRotation;
     D3DXMatrixRotationY(  &matWorld,  timeGetTime()/3000.0f  );
     D3DXMatrixRotationX(  &matRotation,  0.5f  );
     D3DXMatrixMultiply(&matWorld,&matWorld,&matRotation);
          pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

     //  レンダリング
     pDevice->Clear(  0,  NULL,  D3DCLEAR_TARGET  |  D3DCLEAR_ZBUFFER  ,
             D3DCOLOR_XRGB(100,100,100),  1.0f,  0  );

     if(  SUCCEEDED(  pDevice->BeginScene()  )  )
     {
     for(  DWORD  i=0;  i<dwNumMaterials;  i++  )
                 {
                         pDevice->SetMaterial(  &pMeshMaterials[i]  );
                         pDevice->SetTexture(  0,  pMeshTextures[i]  );
                         pMesh->DrawSubset(  i  );
```

```
            }
        pDevice->EndScene();
    }
    pDevice->Present(  NULL,  NULL,  NULL,  NULL  );
```

このコードに到達する前に、pMesh(LPD3DXMESH型)及びマテリアル、テクスチャが初期化されていることが前提です。初期化については直前のTipsを参照してください。

また、言うまでもなくpDeviceも当然初期化されていなくてはなりません。

▼実行結果



OnePoint

DrawSubsetメソッドは“1つのマテリアル”をレンダリングするメソッドです。1つのメッシュに複数のマテリアルがある場合は、その数だけDrawSubsetを実行する必要があります。それがdwNumMaterialsの数（マテリアルの数）でループさせている理由です。

089 メッシュを半透明でレンダリングする

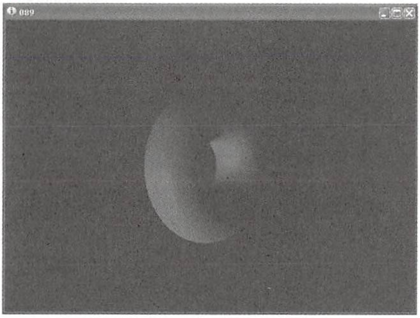
アルファブレンドを有効にする

初級

サンプルプロジェクト　089 メッシュを半透明でレンダリング

IDirect3DDevice9::SetRenderState メソッドでアルファ設定を次のようにします。

▼実行結果



```
IDirect3DDevice9::SetRenderState  ( D3DRS_ALPHABLENDENABLE,  TRUE  );
IDirect3DDevice9::SetRenderState  ( D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA  );
IDirect3DDevice9::SetRenderState  (  D3DRS_DESTBLEND,  D3DBLEND_INVSRCALPHA
);
```

実際の運用は、上の3行をそのまま書けばいいだけです。

サンプルでは、次のようにレンダリングの直前に書いています。

```
pDevice->Clear(        0,        NULL,        D3DCLEAR_TARGET       |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,100),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
{
    //アルファブレンディング設定
    pDevice->SetRenderState  (  D3DRS_ALPHABLENDENABLE,  TRUE  );
    pDevice->SetRenderState  (  D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA  );
    pDevice->SetRenderState  (  D3DRS_DESTBLEND,  D3DBLEND_INVSRCALPHA  );
    //トーラスのレンダリング
    pDevice->SetMaterial(pTorusMeshMaterials);
    pTorusMesh->DrawSubset(0);

    pDevice->EndScene();
}
pDevice->Present(  NULL,  NULL,  NULL,  NULL  );
```

One Point

アルファブレンドを実現する方法はひとつではありません。様々な方法があります。それらについてはTips178以降のアルファブレンディングところで解説します。

090 テクスチャなしメッシュにテクスチャ座標を追加する

●ID3DXMesh::CloneMeshFVFメソッドを使う

上級

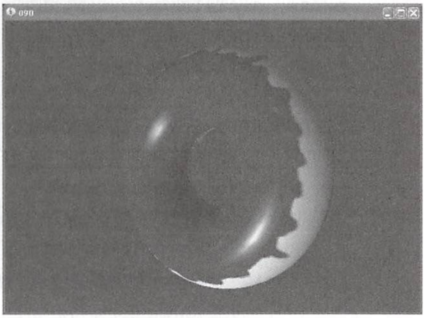
サンプルプロジェクト 090 テクスチャ座標を追加

もともとテクスチャ座標を持たないメッシュ、つまりテクスチャのないメッシュにテクスチャを貼りたいことがあります。

基本形状のメッシュはテクスチャを持たず、また、生成コードも簡単なため、ここではトーラスメッシュにテクスチャを貼ります。

トーラスはドーナツ形状なので、文字通りドーナツ、チョコドーナツにしてみましょう。チョコの部分はテクスチャで表現します。

▼実行結果



```
//トーラスの作成
LPD3DXMESH  pTempMesh=NULL;
D3DXCreateTorus(pDevice,0.5,1,48,48,&pTempMesh,NULL);

//「テクスチャオブジェクト」の作成
if(FAILED(D3DXCreateTextureFromFileEx(pDevice,”Donut.jpg”,256*
(16.0f/24.0f),256*
(16.0f/24.0f),0,0,D3DFMT_UNKNOWN,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAU
LT,0xff000000,NULL,NULL,&pTexture)))
{
    MessageBox(0,”テクスチャの作成に失敗しました”,””,MB_OK);
    return  E_FAIL;
```

この部分はメッシュの作成部分であり、Tips081で解説しています。重要なのは次のコードです。

```
//一時的なトーラスメッシュから、テクスチャ座標を持つメッシュとして複製を作成する
pTempMesh->CloneMeshFVF(NULL,D3DFVF_XYZ | D3DFVF_NORMAL |
D3DFVF_TEX1,pDevice,&pTorusMesh);
SAFE_RELEASE(pTempMesh);
```

初期のトーラスメッシュは、すぐに複製を作成するので一時的な　ものです。そのためpTempMeshとしました。

　複製といってもただ複製するのではなく、もとのメッシュとは異なる構造のものを複製します。もとのメッシュにテクスチャ座標を持つように複製します(; D3DFVF_TEX1)。

```
LPDIRECT3DVERTEXBUFFER9  pVB=NULL;
TEX_VERTEX   ＊pVertices=NULL;
pTorusMesh->GetVertexBuffer(&pVB);
DWORD  NumVertices=pTorusMesh->GetNumVertices();
DWORD  dwStride=pTorusMesh->GetNumBytesPerVertex();
D3DXVECTOR3  vMax,vMin;
FLOAT  fWidth,fHeight;

if(SUCCEEDED(pVB->Lock(0,0,(VOID＊＊)&pVertices,0)))
{
    D3DXComputeBoundingBox(                                          (D3DXVECTOR3
＊)pVertices,NumVertices,dwStride,&vMax,&vMin);
    fWidth=fabs(vMax.x-vMin.x);
    fHeight=fabs(vMax.y-vMin.y);

    for(DWORD  i=0;i<NumVertices;i++)
    {
          if(pVertices[i].Coord.z>-0.1)//表側だけにチョコレートを付けたい！
          {
                pVertices[i].u=vMin.x-(pVertices[i].Coord.x  /  fWidth);
                pVertices[i].v=vMin.y-(pVertices[i].Coord.y  /  fHeight);
          }
          else
          {
                pVertices[i].u=0;
                pVertices[i].v=0;
          }
    }

    pVB->Unlock();
}
```

　この部分のコードは長いですが、何をやっているかと言うと、テクスチャ座標を初期化しています。チョコレートのテクスチャが上手く見えるようになるようにテクスチャ座標を計算しています。具体的には単純なプレーンマッピング（平面状マッピング）です。

　テクスチャ座標を初期化するには頂点バッファをロックし、メッシュ内の頂点にアクセスできるようにする必要があります。

One Point

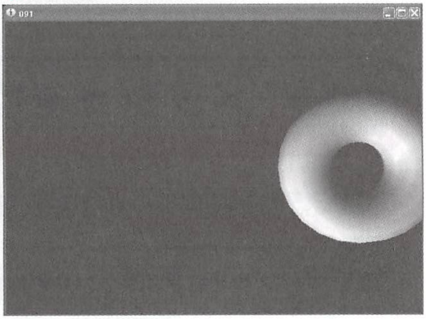
プレーンマッピング以外にも、様々なマッピング形態があり、それぞれ計算方法は様々です。何れにしてもテクスチャの初期化部分のコードが長くなります。

091 メッシュを移動する

●移動行列をもとにワールド変換を行うことにより平行移動する　中級

サンプルプロジェクト　091　メッシュを移動する

▼実行結果



平行移動用の行列を用意して、D3DXMatrixTranslation関数により平行移動行列として初期化します。

D3DXMatrixTranslation関数は、指定されたX,Y,Z3つの座標成分から、平行移動行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した平行移動行列をワールド変換行列に含めて、そのワールド行列をIDirect3DDevice9::SetTransform(D3DTS_WORLD, OOO)メソッドによりワールド変換として登録します。

```
FLOAT  fX=0,fY=0,fZ=0;
```

x,y,z成分の増減を表す変数を用意し、入力によって増減させます。

```
D3DXMATRIX  matWorld,matTrans;
//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(  &matTrans,fX,fY,fZ  );
matWorld*=matTrans;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

pDevice->Clear(  0,  NULL,  D3DCLEAR_TARGET  |  D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB
(0,0,100),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
```

```
{
    //トーラスのレンダリング
    pDevice->SetMaterial(pTorusMeshMaterials);
    pTorusMesh->DrawSubset(0);

    pDevice->EndScene();
}
```

ジオメトリ固有の動き（及び回転、スケーリング等）はワールド変換により制御します。

One Point

ビュートランスフォームとプロジェクショントランスフォームは変化せずに、事前に行っているものとします。

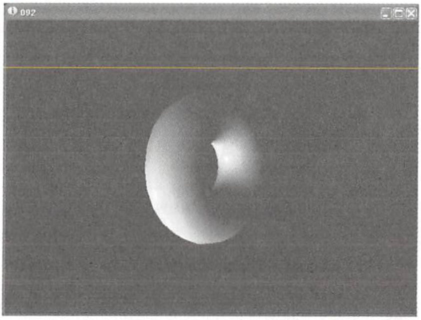
092 メッシュを回転させる

●回転行列を基にワールド変換を行うことにより回転する

中級

サンプルプロジェクト　092　メッシュを回転させる

▼実行結果



回転行列用の行列を用意して、D3DXMatrixRotationX関数、D3DXMatrixRotationY関数、D3DXMatrixRotationZ関数により回転行列として初期化します。

3つの関数は、指定された角度（ラジアン角）から、それぞれX軸、Y軸、Z周りの回転行列を作成し、第1引数に指定した行列インスタンスに代入します。

作成した回転行列をワールド変換行列に含めて、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

```
FLOAT  fYaw=0,fPitch=0,fRoll=0;
```

ヨー回転（Y軸回り）、ピッチ回転（X軸回り）、ロール回転（Z軸回り）を表す変数を用意し、なんらかの入力によって増減させます。

```
D3DXMATRIX  matWorld,matYaw,matPitch,matRoll;
//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixRotationY(  &matYaw,fYaw  );
D3DXMatrixRotationX(  &matPitch,fPitch  );
D3DXMatrixRotationZ(  &matRoll,fRoll   );
matWorld＊=matYaw  ＊matPitch  ＊  matRoll;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

pDevice->Clear(          0,          NULL,          D3DCLEAR_TARGET          |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,100),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
{
    //トーラスのレンダリング
    pDevice->SetMaterial(pTorusMeshMaterials);
    pTorusMesh->DrawSubset(0);

    pDevice->EndScene();
}
```

One Point

ジオメトリ固有の回転（及び平行移動、スケーリング等）はワールド変換により制御します。

One Point

ビュートランスフォームとプロジェクショントランスフォームは変化せずに、事前に行っているものとします。

093 メッシュを拡大縮小する

●スケーリング行列を基にワールド変換を行う

中級

サンプルプロジェクト　093 メッシュを拡大縮小

スケーリング行列用の行列を用意して、D3DXMatrixScaling関数によりスケーリング行列として初期化します。

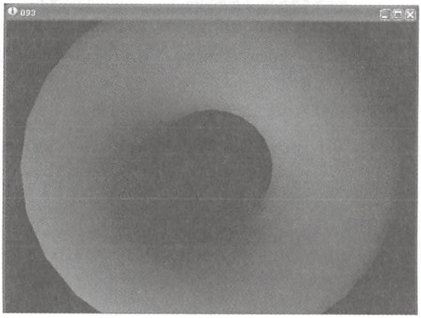
D3DXMatrixScaling関数は、指定された3つの比率から、スケーリング行列を作成し、第1引数に指定した行列インスタンスに代入します。3つの比率は左から順にX,Y,Z軸方向における縮尺です。

作成したスケーリング行列をワールド変換行列に含めて、そのワールド行列をIDirect3DDevice9::SetTransformメソッドによりワールド変換として登録します。

FLOAT fScale=1;

スケールを表す変数を用意し、なんらかの入力によって増減させます。

▼実行結果



```
D3DXMATRIX  matWorld,matScale;
//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixScaling(  &matScale,fScale,fScale,fScale  );
matWorld*=matScale;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );

pDevice->Clear(         0,          NULL,          D3DCLEAR_TARGET    |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,100),  1.0f,  0  );
if(  SUCCEEDED(  pDevice->BeginScene()  )  )
{
    //トーラスのレンダリング
    pDevice->SetMaterial(pTorusMeshMaterials);
    pTorusMesh->DrawSubset(0);

    pDevice->EndScene();
}
```

スケールを表す変数からスケーリング行列を作成し、それをワールド変換に含めます。サンプルでは、その他の変換がないのでたまたま、スケーリング行列＝ワールド行列となっています。

One Point

ビュートランスフォームとプロジェクショントランスフォームは変化せずに、事前に行っているものとします。

094 メッシュをビルボーディングする

●ビュー行列の逆行列によりビルボーディングをかける　上級

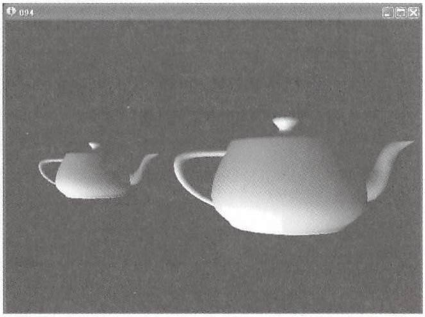
サンプルプロジェクト　094　メッシュをビルボーディング

図において、右側のプリミティブがビルボーディングをかけたメッシュです。板状のジオメトリにビルボーディングをかけたものを特にビルボード（広告・看板）と呼びますが、立体的なジオメトリであるメッシュもビルボードになると思います。そう呼ぶかどうかは別として、立体的なメッシュにビルボーディングをかけたい場合は往々にしてあります。

本サンプルは矢印キーでカメラ(視点)が回転するようになっています。図の状態で左側のプリミティブはビルボーディングを書けていない通常のものなので、当然回転します。それに対し、右のビルボードメッシュは、視点がどうであろうと常に同じ姿勢に留まります。

ビルボーディングは、その時点のビュー変換の逆行列をかけることにより実現します。それにより、それ以降のレンダリングは、カメラの影響を受けません。

▼実行結果



```
//1個目のティーポットのレンダリング（通常）
pDevice->SetMaterial(pTeapotMeshMaterials);
pTeapotMesh->DrawSubset(0);

//2個目のティーポットのレンダリング（ビルボーディングをかける）
D3DXMATRIX  matBillboard;
D3DXMatrixInverse(&matBillboard,NULL,&matView);//ビルボーディング用打ち消し行列作成
```

```
matWorld＊=matBillboard;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->SetMaterial(pTeapotMeshMaterials);
pTeapotMesh->DrawSubset(0);
```

One Point

実際的に言うと、カメラの影響を受けないのではなく、カメラの影響を受けてちょうど元に戻るような変換がなされます。結果的に影響を受けていないように見えるということです。

095 1つのメッシュを複数レンダリングする

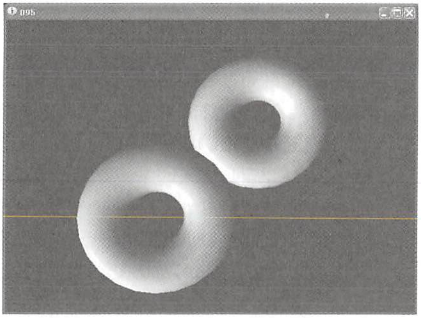
●1つのインスタンスを異なる座標にレンダリングして複数に見せる　中級

サンプルプロジェクト　095　1つのメッシュを複数レンダリング

メッシュの実態そのものは単一のもの、つまり単一のインスタンスに、複数の座標を関連付けて、それぞれの座標にレンダリングすれば複数のオブジェクトに見せることができます。

それぞれのレンダリングには、それぞれのワールド変換を行います。

▼実行結果



```
//1個目のレンダリング
//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(  &matTrans,-1,-1,0  );
matWorld＊=matTrans;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->SetMaterial(pTorusMeshMaterials);
pTorusMesh->DrawSubset(0);

//2個目のレンダリング
```

```
//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(  &matTrans,1,1,1 );
matWorld*=matTrans;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->SetMaterial(pTorusMeshMaterials);
pTorusMesh->DrawSubset(0);
```

One Point

アニメーションメッシュにおいては、全てが同じモーションになってしまうので、インスタンス自体を固有に持たせなければなりません。複数のインスタンスについてはTips096のサンプルを参考にしてください。

096 複数のメッシュをレンダリングする

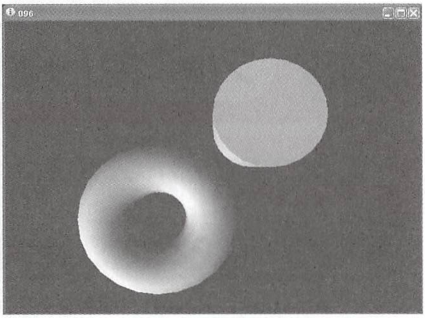
●メッシュのインスタンス自体を個々に用意する

中級

サンプルプロジェクト　096　複数のメッシュをレンダリング

インスタンスをそれぞれ作成します。ここでは、分かりやすいように異なる形状を作成しました。

▼実行結果



```
//トーラスの作成
    D3DXCreateTorus(pDevice,0.5,1,12,24,&pTorusMesh,NULL);
//シリンダーの作成
    D3DXCreateCylinder(pDevice,1,1,3,24,5,&pCylinderMesh,NULL);
```

```
//トーラスのレンダリング

//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(  &matTrans,-1,-1,0  );
matWorld＊=matTrans;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->SetMaterial(pTorusMeshMaterials);
pTorusMesh->DrawSubset(0);

//シリンダーのレンダリング

//ワールドトランスフォーム（絶対座標変換）
D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(  &matTrans,1,1,1  );
matWorld＊=matTrans;
pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
pDevice->SetMaterial(pCylinderMeshMaterials);
pCylinderMesh->DrawSubset(0);
```

One Point

アニメーションメッシュや、形状の異なる“絵”（メッシュ等）をもつキャラクターをレンダリングする場合は、当然それぞれについて、インスタンスを持たせなければなりません。同じ“絵”の使い回しはできないということです。

097 フルスクリーンに復帰した時に再初期化をする

●デバイスをリセットし、同時に、必要なオブジェクトを再度初期化する　初級

サンプルプロジェクト　097　フルスクリーンからの復帰

Alt+tabキーにより、画面をデスクトップに切り替えると、フルスクリーンでのリソースは破壊されます。破壊されたリソース（ここではメッシュオブジェクト）は元に戻さなくてはなりません。（再生成しなくてはなりません）このことの本質については、Tips020を参照してください。

デバイスを失うと、レンダリングステートはリセットされてしまいます。なので、フルスクリーンに復帰した時は、レンダリングステートを再度設定しなくてはなりません。そして、トランスフォーム（ビュー、プロジェクション）も設定し直す必要があります。

```
//VOID  Render()
//レンダリングする関数
VOID  Render()
{
    D3DXMATRIX  matWorld;
    D3DXMATRIX  matView;
    D3DXMATRIX  matProj;
    //ワールドトランスフォーム（絶対座標変換）
    D3DXMatrixRotationY(  &matWorld,  timeGetTime()/1000.0f  );
    pDevice->SetTransform(  D3DTS_WORLD,  &matWorld  );
    //  ビュートランスフォーム（視点座標変換）
    D3DXVECTOR3  vecEyePt(  0.0f,  0.0f,-3.0f  );  //カメラ（視点）位置
    D3DXVECTOR3  vecLookatPt(  0.0f,  0.0f,  0.0f  );//注視位置
    D3DXVECTOR3  vecUpVec(  0.0f,  1.0f,  0.0f  );//上方位置
    D3DXMatrixLookAtLH(  &matView,  &vecEyePt,  &vecLookatPt,  &vecUpVec  );
    pDevice->SetTransform(  D3DTS_VIEW,  &matView  );
    //  プロジェクショントランスフォーム（射影変換）
    D3DXMatrixPerspectiveFovLH(          &matProj,          D3DX_PI/4,
(FLOAT)WINDOW_WIDTH/(FLOAT)WINDOW_HEIGHT,  0.1f,  1000.0f  );
    pDevice->SetTransform(  D3DTS_PROJECTION,  &matProj  );


    pDevice->Clear(        0,          NULL,        D3DCLEAR_TARGET          |
D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(30,30,30),  1.0f,  0  );
    if(  SUCCEEDED(  pDevice->BeginScene()  )  )
    {
        //スフィアのレンダリング
        pDevice->SetMaterial(pSphereMeshMaterials);
        pSphereMesh->DrawSubset(0);

        pDevice->EndScene();
    }

    //デバイス消失への対応
    if(pDevice->Present(  NULL,  NULL,  NULL,  NULL  )==D3DERR_DEVICELOST)
    {
        if(pDevice->TestCooperativeLevel()  ==  D3DERR_DEVICELOST)
        {
            return;
        }

        SAFE_RELEASE(  pSphereMesh  );

        D3DPRESENT_PARAMETERS  d3dpp;
```

```
          ZeroMemory(  &d3dpp,  sizeof(d3dpp)  );
          D3DDISPLAYMODE  DMode;
          pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT  ,&DMode);
          d3dpp.BackBufferFormat  =DMode.Format;
          d3dpp.BackBufferWidth=WINDOW_WIDTH;
          d3dpp.BackBufferHeight=WINDOW_HEIGHT;
          d3dpp.BackBufferCount=1;
          d3dpp.SwapEffect  =  D3DSWAPEFFECT_DISCARD;
          d3dpp.Windowed  =  FALSE;
          d3dpp.EnableAutoDepthStencil  =  TRUE;
          d3dpp.AutoDepthStencilFormat  =  D3DFMT_D16;

          if(pDevice->Reset(&d3dpp)!=D3D_OK)
          {
                MessageBox(0,”デバイスのリセットに失敗しました”,"",MB_OK);
          }
          //スフィアの”再”初期化
          D3DXCreateSphere(pDevice,1,24,24,&pSphereMesh,NULL);
          //  ライトを”再度”オンにする
          pDevice->SetRenderState(  D3DRS_LIGHTING,  TRUE  );
          //  ライトを”再度”あてる  白色で光沢ありに設定
            D3DXVECTOR3  vecDirection(1,1,1);
            D3DLIGHT9  light;
            ZeroMemory(  &light,  sizeof(D3DLIGHT9)  );
            light.Type              =  D3DLIGHT_DIRECTIONAL;
            light.Diffuse.r     =  1.0f;
            light.Diffuse.g     =  1.0f;
            light.Diffuse.b     =  1.0f;
            light.Specular.r=1.0f;
            light.Specular.g=1.0f;
            light.Specular.b=1.0f;
                D3DXVec3Normalize(     (D3DXVECTOR3＊   )&light.Direction,
&vecDirection  );
            light.Range               =  200.0f;
            pDevice->SetLight(  0,  &light  );
            pDevice->LightEnable(  0,  TRUE  );
    }
}
```

098 プログレッシブメッシュをレンダリングする

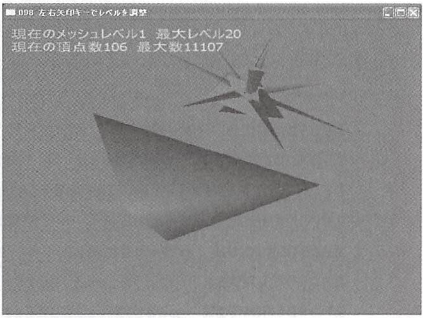
●ID3DXPMesh::DrawSubsetメソッドを使う

上級

サンプルプロジェクト　098 プログレッシブメッシュをレンダリング

最低レベル（最も少ないポリゴン数）のメッシュをレンダリングしたところ。“へた”の部分はこれでもいい感じですが、“果実“の部分が形を成してません。

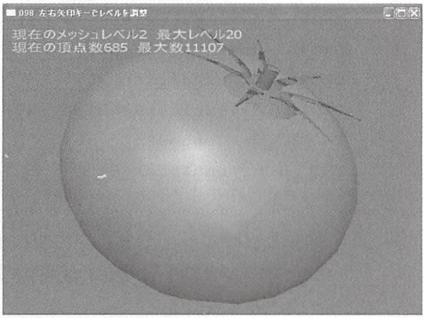
▼実行結果


098 左右矢印キーでレベルを調整

```
現在のメッシュレベル1  最大レベル20
現在の頂点数106  最大数11107
```

下から2番目に少ないポリゴン数のメッシュです。ぱっと見ると最高レベルのメッシュと見分けがつきません。全てこのレベルのメッシュでいってもいいくらいです。

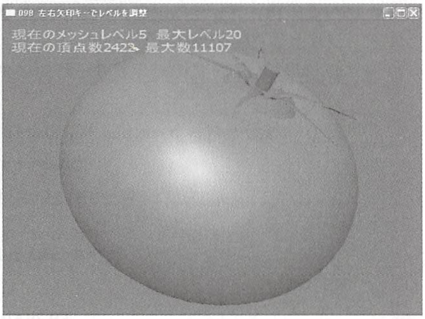
▼実行結果


098 左右矢印キーでレベルを調整

```
現在のメッシュレベル2  最大レベル20
現在の頂点数685  最大数11107
```

レベル5のメッシュになると、最高レベルのメッシュと見分けがつきません。

▼実行結果

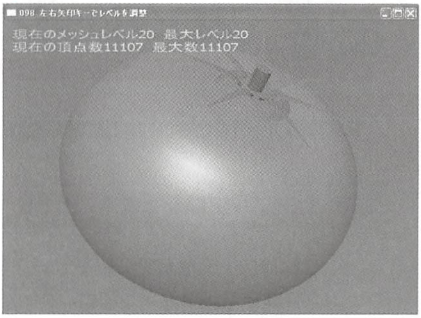

098 左右矢印キーでレベルを調整

```
現在のメッシュレベル5  最大レベル20
現在の頂点数2422  最大数11107
```

最高レベルのメッシュは、見栄えが変わらないにも関らず、頂点・ポリゴン数が非常に多いので無駄と言えます。

ただし、このトマトメッシュについてであって、メッシュによっては見分けが付くものがあるかもしれません。

▼実行結果



098 左右矢印キーでレベルを調整

現在のメッシュレベル20　最大レベル20
現在の頂点数11107　最大数11107

```
LPD3DXMESH  pMesh  =  NULL;
LPD3DXPMESH＊  ppPGMesh  =  NULL;
D3DMATERIAL9＊  pMeshMaterials  =  NULL;
LPD3DXMESH  pTempMesh  =  NULL;
LPD3DXPMESH  pTempPGMesh  =  NULL;
LPD3DXBUFFER  pAdjacencyBuffer  =  NULL;
LPD3DXBUFFER  pD3DXMtrlBuffer  =  NULL;
LPDIRECT3DTEXTURE9＊  pMeshTextures    =  NULL;
DWORD  dwNumMaterials  =  0;
DWORD  dwMinVertices=0;
DWORD  dwMaxVertices=0;
DWORD  dwVerticesPerMesh  =0;
DWORD  dwPMeshesAmt  =0;
INT  iCurrentIndex=0;
```

メッシュオブジェクト。プログレッシブメッシュオブジェクト、その他必要な変数をグローバルに宣言しています。

```
//  xファイルからメッシュをロードする　。
//この後、クリーニング、レベルごとの分割メッシュの作成という処理を行うが、その都度、新たなメッシュとして再構築される。
//したがって、この時点では一時的なメッシュなので、一時的なポインターを用意する

if(  FAILED(  D3DXLoadMeshFromX(   “tomato.x”  ,  D3DXMESH_SYSTEMMEM,
                                          pDevice,  &pAdjacencyBuffer,  &pD3DXMtrlBuffer,  NULL,
                                 &dwNumMaterials,  &pTempMesh  )  )  )
              {
                                          MessageBox(NULL,    “xファイルの読み込みに失敗しました”,NULL,
MB_OK);
                                            return  E_FAIL;
```

```
        }

D3DXMATERIAL＊      d3dxMaterials      =      (D3DXMATERIAL＊ )pD3DXMtrlBuffer-
>GetBufferPointer();
        pMeshMaterials   =  new  D3DMATERIAL9[dwNumMaterials];
        pMeshTextures     =  new  LPDIRECT3DTEXTURE9[dwNumMaterials];

for(  DWORD  i=0;  i<dwNumMaterials;  i++  )
{
    pMeshMaterials[i]  =  d3dxMaterials[i].MatD3D;
                pMeshMaterials[i].Ambient  =  pMeshMaterials[i].Diffuse;
                pMeshTextures[i]  =  NULL;
                if(  d3dxMaterials[i].pTextureFilename  !=  NULL  &&
                        lstrlen(d3dxMaterials[i].pTextureFilename)  >  0  )
                {
                        if(  FAILED(  D3DXCreateTextureFromFile(  pDevice,

d3dxMaterials[i].pTextureFilename,

&pMeshTextures[i]  )  )  )
                        {
                                MessageBox(NULL,  “テクスチャの読み込みに失敗しまし
た”,  NULL,  MB_OK);
                }
                }
        }
    pD3DXMtrlBuffer->Release();
```

この部分は、他のTipsでも解説している、普通のメッシュのロードルーチンです。次からが、プログレッシブメッシュ特有のものです。

コードが長いので、処理の流れを簡潔に書くと次のようになります。

1．まず最初に、クリーニングを行います。クリーニングとは、メッシュを単純化するために前処理です。
2．無駄な頂点を結合します。
3．無効なインデックスがないかどうかメッシュを検証します。
4．プログレッシブメッシュオブジェクトを作成します。
5．プログレッシブメッシュを構成する分割メッシュを作成します。

1．クリーニング

```
//クリーニング（最適化・単純化するための前処理）
if(FAILED(D3DXCleanMesh(D3DXCLEAN_SIMPLIFICATION,pTempMesh,(DWORD
＊)pAdjacencyBuffer->GetBufferPointer(),
    &pMesh,(DWORD＊)pAdjacencyBuffer->GetBufferPointer(),  NULL)))
{
      MessageBox(NULL,   “メッシュのクリーニングに失敗しました”,  NULL,  MB_OK);
      return  E_FAIL;
}
```

2. 頂点結合

```
//無駄な頂点を結合
          if(  FAILED(  D3DXWeldVertices(  pMesh,  0,  NULL,
                              (DWORD＊)pAdjacencyBuffer->GetBufferPointer(),
                              (DWORD＊)pAdjacencyBuffer->GetBufferPointer(),  NULL,
NULL  )  )  )
{
          MessageBox(NULL,   “頂点の結合に失敗しました”,  NULL,  MB_OK);
       return  E_FAIL;
}
```

3. 検証

```
//メッシュの検証（無効なインデックスの有無をチェック）
if(    FAILED(    D3DXValidMesh(    pMesh,    (DWORD＊  )pAdjacencyBuffer-
>GetBufferPointer(),  NULL  )  )  )
{
             MessageBox(NULL,   “メッシュの検証に失敗しました”,  NULL,  MB_OK);
          return  E_FAIL;
}
```

4. プログレッシブメッシュオブジェクト作成

```
//プログレッシブメッシュとして再構築
if(FAILED(D3DXGeneratePMesh(    pMesh,    (DWORD＊   )pAdjacencyBuffer-
>GetBufferPointer(),
                              NULL,  NULL,  1,  D3DXMESHSIMP_VERTEX,
&pTempPGMesh  )))
{
      MessageBox(NULL,   “プログレッシブメッシュの作成に失敗しました”,  NULL,  MB_OK);
      return  E_FAIL;
}
```

5. 分割メッシュ作成

```
//分割メッシュを作成
dwMinVertices=pTempPGMesh->GetMinVertices();
```

```
dwMaxVertices=pTempPGMesh->GetMaxVertices();
dwVerticesPerMesh = ( dwMaxVertices - dwMinVertices + SEPARATE_LEVELS
) / SEPARATE_LEVELS;
dwPMeshesAmt = max( 1, (DWORD)ceil( (dwMaxVertices - dwMinVertices +
1) / (FLOAT)dwVerticesPerMesh ) );

ppPGMesh=new LPD3DXPMESH[dwPMeshesAmt];
DWORD d=pTempPGMesh->GetFVF();
for(DWORD i=0;i<dwPMeshesAmt;i++)
{
    if(FAILED(pTempPGMesh->ClonePMeshFVF(          D3DXMESH_MANAGED          |
D3DXMESH_VB_SHARE,
            pTempPGMesh->GetFVF(), pDevice, &ppPGMesh[ i ] )))
    {
            MessageBox(NULL, “分割メッシュのクローンに失敗しました”, NULL, MB_OK);
            return E_FAIL;
    }
    if(FAILED(ppPGMesh[ i ]->TrimByVertices(dwMinVertices+dwVerticesPerMesh
＊i,
            dwMinVertices+dwVerticesPerMesh＊(i+1),NULL,NULL)))
    {
            MessageBox(NULL,    “分割メッシュごとのトリムに失敗しました”,    NULL,
MB_OK);
            return E_FAIL;
    }
    if(FAILED(ppPGMesh[ i ]->OptimizeBaseLOD( D3DXMESHOPT_VERTEXCACHE, NULL
)))
    {
            MessageBox(NULL,    “分割メッシュごとの最適化に失敗しました”,    NULL,
MB_OK);
            return E_FAIL;
    }
}
```

あとは、次のように通常のメッシュと同じようにレンダリングします。

```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    for( DWORD i=0; i<dwNumMaterials; i++ )
    {
                pDevice->SetMaterial( &pMeshMaterials[i] );
                pDevice->SetTexture( 0, pMeshTextures[i] );
                ppPGMesh[iCurrentIndex]->DrawSubset(  i  );
        }
      pDevice->EndScene();
    }
```

One Point

Tips098はコードが長いですが、長いのは初期化コード部分であり、Direct3Dの仕様上の問題ですから、無理に理解しようとすることはないと思います。初期化が完了すれば、通常のメッシュとほぼ同じように扱えます。

099 プログレッシブメッシュを作成する

段階的にポリゴン数を落としたメッシュを複数作成する

上級

参考プロジェクト　098　プログレッシブメッシュをレンダリング

プログレッシブメッシュのレンダリングのコードは、例えば次のようになります。

```
LPD3DXMESH  pMesh  =  NULL;
LPD3DXPMESH*  ppPGMesh  =  NULL;
D3DMATERIAL9*  pMeshMaterials  =  NULL;
LPD3DXMESH  pTempMesh  =  NULL;
LPD3DXPMESH  pTempPGMesh  =  NULL;
LPD3DXBUFFER  pAdjacencyBuffer  =  NULL;
LPD3DXBUFFER  pD3DXMtrlBuffer  =  NULL;
LPDIRECT3DTEXTURE9*  pMeshTextures    =  NULL;
DWORD  dwNumMaterials  =  0;
DWORD  dwMinVertices=0;
DWORD  dwMaxVertices=0;
DWORD  dwVerticesPerMesh  =0;
DWORD  dwPMeshesAmt  =0;
INT  iCurrentIndex=0;

//  xファイルからメッシュをロードする　。
//この後、クリーニング、レベルごとの分割メッシュの作成という処理を行うが、その都度、新たなメッシュとして再構築される。
```

```
//したがって、この時点では一時的なメッシュなので、一時的なポインターを用意する

if(  FAILED(  D3DXLoadMeshFromX(   “tomato.x” ,  D3DXMESH_SYSTEMMEM,
                      pDevice,  &pAdjacencyBuffer,  &pD3DXMtrlBuffer,  NULL,
                 &dwNumMaterials,  &pTempMesh   )   )   )
       {
                      MessageBox(NULL,    “xファイルの読み込みに失敗しました”,NULL,
MB_OK);
                       return  E_FAIL;
       }

D3DXMATERIAL＊     d3dxMaterials    =    (D3DXMATERIAL＊ )pD3DXMtrlBuffer-
>GetBufferPointer();
       pMeshMaterials  =  new  D3DMATERIAL9[dwNumMaterials];
       pMeshTextures    =  new  LPDIRECT3DTEXTURE9[dwNumMaterials];

for(  DWORD  i=0;  i<dwNumMaterials;  i++  )
{
    pMeshMaterials[i]  =  d3dxMaterials[i].MatD3D;
                pMeshMaterials[i].Ambient  =  pMeshMaterials[i].Diffuse;
                pMeshTextures[i]  =  NULL;
                if(  d3dxMaterials[i].pTextureFilename  !=  NULL  &&
                       lstrlen(d3dxMaterials[i].pTextureFilename)  >  0  )
                {
                       if(  FAILED(  D3DXCreateTextureFromFile(  pDevice,

d3dxMaterials[i].pTextureFilename,

&pMeshTextures[i]  )  )  )
                       {
                              MessageBox(NULL,    “テクスチャの読み込みに失敗しまし
た”,  NULL,  MB_OK);
               }
                }
       }
    pD3DXMtrlBuffer->Release();
```

コードが長いので、処理の流れを簡潔に書くと次のようになります。

1. まず最初に、クリーニングを行います。クリーニングとは、メッシュを単純化するために前処理です。
2. 無駄な頂点を結合します。
3. 無効なインデックスがないかどうかメッシュを検証します。
4. プログレッシブメッシュオブジェクトを作成します。
5. プログレッシブメッシュを構成する分割メッシュを作成します。

1. クリーニング

```
//クリーニング（最適化・単純化するための前処理）
if(FAILED(D3DXCleanMesh(D3DXCLEAN_SIMPLIFICATION,pTempMesh,(DWORD
＊)pAdjacencyBuffer->GetBufferPointer(),
    &pMesh,(DWORD＊)pAdjacencyBuffer->GetBufferPointer(),  NULL)))
{
    MessageBox(NULL,  “メッシュのクリーニングに失敗しました”,  NULL,  MB_OK);
    return  E_FAIL;
}
```

2. 頂点結合

```
//無駄な頂点を結合
        if(  FAILED(  D3DXWeldVertices(  pMesh,  0,  NULL,
                            (DWORD＊)pAdjacencyBuffer->GetBufferPointer(),
                            (DWORD＊)pAdjacencyBuffer->GetBufferPointer(),  NULL,
NULL  )  )  )
{
        MessageBox(NULL,  “頂点の結合に失敗しました”,  NULL,  MB_OK);
     return  E_FAIL;
```

3. 検証

```
//メッシュの検証（無効なインデックスの有無をチェック）
if(     FAILED(     D3DXValidMesh(     pMesh,     (DWORD＊  )pAdjacencyBuffer-
>GetBufferPointer(),  NULL  )  )  )
{
            MessageBox(NULL,  “メッシュの検証に失敗しました”,  NULL,  MB_OK);
        return  E_FAIL;
}
```

4. プログレッシブメッシュオブジェクト作成

```
//プログレッシブメッシュとして再構築
if(FAILED(D3DXGeneratePMesh(       pMesh,       (DWORD＊    )pAdjacencyBuffer-
>GetBufferPointer(),
                                          NULL,  NULL,  1,  D3DXMESHSIMP_VERTEX,
&pTempPGMesh  )))
{
    MessageBox(NULL,  “プログレッシブメッシュの作成に失敗しました”,  NULL,  MB_OK);
    return  E_FAIL;
}
```

5. 分割メッシュ作成

```
//分割メッシュを作成
dwMinVertices=pTempPGMesh->GetMinVertices();
dwMaxVertices=pTempPGMesh->GetMaxVertices();
dwVerticesPerMesh = ( dwMaxVertices - dwMinVertices + SEPARATE_LEVELS
) / SEPARATE_LEVELS;
dwPMeshesAmt = max( 1, (DWORD)ceil( (dwMaxVertices - dwMinVertices +
1) / (FLOAT)dwVerticesPerMesh ) );

ppPGMesh=new LPD3DXPMESH[dwPMeshesAmt];

DWORD d=pTempPGMesh->GetFVF();
for(DWORD i=0;i<dwPMeshesAmt;i++)
{
    if(FAILED(pTempPGMesh->ClonePMeshFVF(      D3DXMESH_MANAGED       |
D3DXMESH_VB_SHARE,
            pTempPGMesh->GetFVF(), pDevice, &ppPGMesh[ i ]  )))
    {
            MessageBox(NULL, “分割メッシュのクローンに失敗しました”, NULL, MB_OK);
            return E_FAIL;
    }
    if(FAILED(ppPGMesh[ i ]->TrimByVertices(dwMinVertices+dwVerticesPerMesh
＊i,
            dwMinVertices+dwVerticesPerMesh＊(i+1),NULL,NULL)))
    {
            MessageBox(NULL,   “分割メッシュごとのトリムに失敗しました”,   NULL,
MB_OK);
            return E_FAIL;
    }
    if(FAILED(ppPGMesh[ i ]->OptimizeBaseLOD( D3DXMESHOPT_VERTEXCACHE, NULL
)))
    {
            MessageBox(NULL,   “分割メッシュごとの最適化に失敗しました”,   NULL,
MB_OK);
            return E_FAIL;
    }
}
```

One Point

コードが長いですが、初期化コードなので固定的です。つまりコピー＆ペーストで利用できます。

アニメーションメッシュ

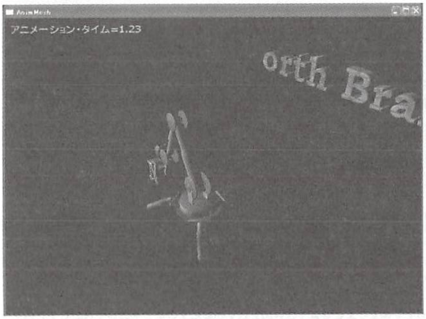
100 アニメーションメッシュとは

●アニメーションメッシュの意味を把握する

上級

参考サンプルプロジェクト　105 アニメーションメッシュ

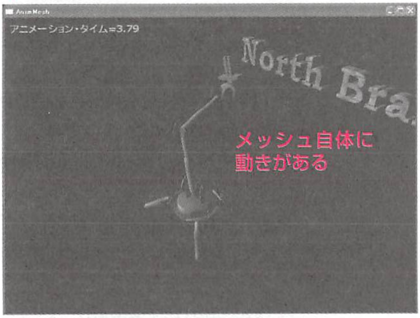
▼実行結果



アニメーション・タイム=1.23

AnimMesh
orth Bra

▼実行結果



アニメーション・タイム=3.79

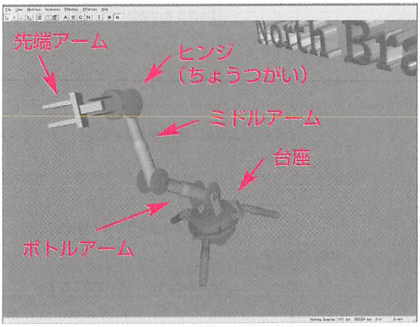
AnimMesh
North Bra
メッシュ自体に
動きがある

アニメーションメッシュとは、文字通りそれ自体に動きのあるメッシュのことです。通常のメッシュももちろん行列により動かすことはできますが（Tips091〜093）“メッシュそのもの“に動きはありません。

アニメーションメッシュでは、複数のメッシュを一まとまりのジオメトリとなります。ですので、Xファイルとしても1つのXファイルに収録され、ファイル名を見ただけでは通常のメッシュなのかアニメーションメッシュなのかは判断できず、実際にレンダリングしてみなければ分かりません。

アニメーションメッシュでは、内部であたかも複数のメッシュを連動させて動かしアニメーションを実現するのですが、単一のメッシュでもメッシュの頂点自体を動かしてアニメーションを実現させる方法もあります。それは特にスキンアニメーションメッシュと言います。

▼これで1つのアニメーション“メッシュ”（RobotArm.x）



North Bra
先端アーム
ヒンジ
（ちょうつがい）
ミドルアーム
台座
ボトルアーム

スキンメッシュについては、Tips109以降で解説しています。

さて、ロボットアームを例にすると、台座、支柱、先端アーム部分…などの各パーツが、言ってみればパーツメッシュということです。現実と同じようにパーツ別にメッシュを分け、

それぞれに動きを与えることができます。

ただし、それぞれに独立した動きを与えることはできますが、独立して動かすのであれば、通常のメッシュとしてパーツごとに読み込んでシーン上でそれらひとつひとつを動かすのと変わりはないので、“うまみ”がありません。

アニメーションメッシュの場合、各パーツメッシュは、階層（Hierarchy：ヒエラルキー）関係にあります。

階層とは、親子関係及び兄弟関係を設定することです。親メッシュの動きや姿勢は子メッシュに影響を与え、親メッシュの動きに連動して子メッシュも動きます。兄弟同士は影響し合いません。階層にする理由は、メッシュの姿勢を相対的にすることであり、相対姿勢にすることにより効率的にアニメーションが実現できるからです。

例えば、ロボットアーム全体が回転する場合、階層関係ではなく各メッシュが独立していると、全てのパーツについて回転を掛けなくてはなりません。また、ロボットアーム全体の回転と同時に先端アームを独自に回転させたい場合、階層関係にないと先端アームの回転は全体の回転を含めて計算しなくてはなりません。

階層関係にある場合、先端アームを支柱の子に設定しておけば、先端アームは先端アーム自身の回転をかけてやるだけで支柱の回転と合成された回転をしてくれます。

One Point

アニメーションメッシュの基本についての詳細は、工学社発行の拙書『はじめての3Dゲーム開発』で解説しています。

101 アニメーションメッシュの作成について

3DCGソフトでの作成過程を見る

中級

Tips074「メッシュの作成について」でも述べましたが、ここで言う“作成”とは、外部ファイル等による静的な作成のことです。つまり素材としての作成です。素材作成の段階はDirectXの範囲外ですが、けっこう重要なことだと思うので記しておきます。

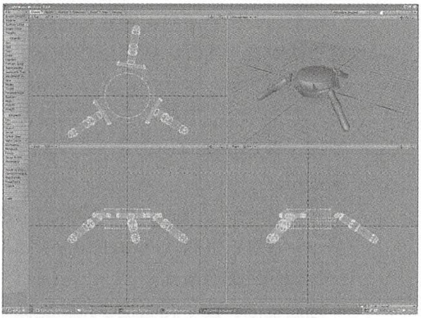
アニメーションメッシュとなると、もはやコード中での動的な作成は現実的ではありませんし、そのような素材生成関数も用意されていません。（もちろんアニメーションメッシュを読み込んだり、操作する関数はあります）作成は通常のメッシュと同じく、3DCGソフトを使用します。

ここでは、筆者が使用しているLightwave 3Dという3DCGソフトを例に解説します。

次の図はLightWaveでモデリングし、Xファイルに出力する過程です。全体のアニメーションオブジェクトを構成する各パーツオブジェクトの作成は、通常の単一メッシュと同じです。異なるのは、“アニメーション付け（モーション付け）”という工程が加わる部分です。

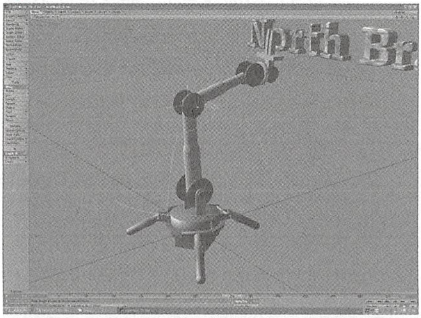
まず、個々のパーツをモデリングします。図では土台のパーツだけしか映ってませんが、その他のパーツも“1つのオブジェクト、メッシュ”として作成します。

▼モデリング



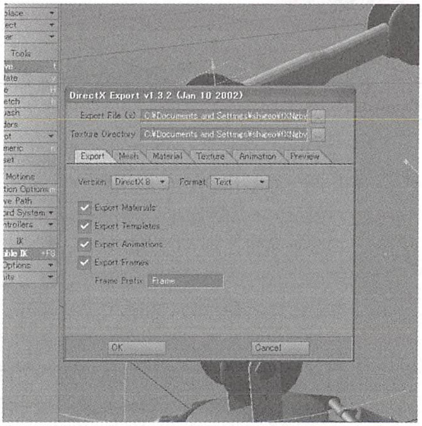
次に、それらを関連付けて、動き（モーション）を付けます。この部分がアニメーションメッシュ作成の固有部分でしょう。

▼モーション付け



ここまでできれば、あとは通常のメッシュの場合と同じように、Xファイルコンバーターでxファイルにエクスポートします。

▼Xファイルとして出力



DirectX Export v1.3.2 (Jan 10 2002)

Export File (x) C:¥Documents and Settings¥shigeo¥DXNgby

Texture Directory C:¥Documents and Settings¥shigeo¥DXNgby

Export
Mesh
Material
Texture
Animation
Preview

Version DirectX 8

Format Text

☑ Export Materials

☑ Export Templates

☑ Export Animations

☑ Export Frames

Frame Prefix Frame

OK
Cancel

OnePoint

3DCGソフトの購入を考えている場合は、　います。
Tips028とコラムが多少なりとも参考になると思

102 アニメーションメッシュをXファイルから読み込む

D3DXLoadMeshHierarchyFromX関数を使う

上級

参考サンプルプロジェクト　105 アニメーションメッシュ

アニメーションメッシュの読み込みは、通常のメッシュと異なり、簡単な関数実行ではできません。

まず、アロケートヒエラルキークラスをアプリケーションで実装します。次にアロケートヒエラルキークラスのインスタンスを作成し、そのインスタンスを引数にしてようやくD3DXLoadMeshHierarchyFromX関数を実行できます。

このアロケートヒエラルキークラスの実装がかなり面倒です。しかし、面倒ではあるものの初期化コードなので固定的であり、一旦書いてしまえば、使いまわしができます。面倒なことは考えずに、単純にコピーアンドペーストするのもいいと思います。極めて仕様上からくる複雑性ですし、深く考えるのが有益なのかどうかは疑問です。

サンプルでは、アロケートヒエラルキークラスの実装はHierarchy.cppにまとめています。AnimMesh.cppでは、次のように簡単にアロケートヒエラルキークラスのインスタンスを作成できます。

```
MY_HIERARCHY cHierarchy;
```

そして、D3DXLoadMeshHierarchyFromX関数によりアニメーションメッシュを読み込みます。

```
if( FAILED(
D3DXLoadMeshHierarchyFromX(szXFileName, D3DXMESH_MANAGED, pDevice,
&cHierarchy,
                                        NULL, &pThing->pFrameRoot, &pThing-
>pAnimController)))
{
        MessageBox(NULL, "Xファイルの読み込みに失敗しました",szXFileName, MB_OK);
        return E_FAIL;
}
```

pThingは、THING構造体のポインターです。このサンプルではTHINGという構造体を作成し、そこにすべてを詰め込んでいます。変数名から想像できると思いますが、第4引数はフレームを意味し、第5引数はアニメーションコントローラーです。

第1引数はXファイル名ですので、D3DXLoadMesh HierarchyFromX(”RobotArm.x” ,と書いても構いません。

103 フレーム階層を展開・走査する

再帰コードで各パーツメッシュをレンダリングする

上級

参考サンプルプロジェクト　105 アニメーションメッシュ

アニメーションメッシュの場合、階層構造であるがゆえに、何ををやるにも階層の走査あるいは展開をしなくてはなりません。

階層の走査・展開は再帰コードを書くことにより簡単にできます。再帰による階層走査は、フレームをレンダリングするDrawFrame関数と、各フレームのワールド変換を行うUpdateFrameMatrices関数を眺めると見えてきます。

2つの関数は、共に似たようなコードであり、共通した構造になっているのが分るでしょうか？

pFrameSiblingとpFrameFirstChildに何らかのアドレスが入っているかどうかで、再帰させています。これにより、階層をどんどん展開し、各ノード中で何らかの処理を行うので、結局全てのノードを処理することになります。

```
//
//VOID DrawFrame(LPD3DXFRAME pFrame)
//複数のメッシュから成るフレームをレンダリングする。
VOID DrawFrame(LPD3DXFRAME pFrame)
{

    LPD3DXMESHCONTAINER pMeshContainer;

    pMeshContainer = pFrame->pMeshContainer;
    while (pMeshContainer != NULL)
    {
        RenderMeshContainer(pMeshContainer, pFrame);

        pMeshContainer = pMeshContainer->pNextMeshContainer;
    }

    if (pFrame->pFrameSibling != NULL)
    {
        DrawFrame(pFrame->pFrameSibling);
    }

    if (pFrame->pFrameFirstChild != NULL)
    {
        DrawFrame(pFrame->pFrameFirstChild);
    }
}


//
//VOID UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX
pParentMatrix)
//フレーム内のメッシュ毎にワールド変換行列を更新する
VOID UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX pParentMatrix)
{
    MYFRAME ＊pFrame = (MYFRAME＊)pFrameBase;

    if (pParentMatrix != NULL)
    {
        D3DXMatrixMultiply(&pFrame->CombinedTransformationMatrix, &pFrame-
>TransformationMatrix, pParentMatrix);
    }
    else
    {
        pFrame->CombinedTransformationMatrix = pFrame->TransformationMatrix;
    }

    if (pFrame->pFrameSibling != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameSibling, pParentMatrix);
    }

    if (pFrame->pFrameFirstChild != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameFirstChild, &pFrame-
>CombinedTransformationMatrix);
    }
}
```

One Point

アニメーションメッシュの基本についての詳細は、工学社発行の拙書『はじめての3Dゲーム開発』で解説しています。

104 階層展開(アロケート・ヒエラルキー)クラスのメソッドを実装する

仮想関数になっている4つの関数を実装する

上級

参考サンプルプロジェクト　105　アニメーションメッシュ

アロケートヒエラルキークラスの実装はかなりの分量です。

ポイントは、4つのメソッドの実装です。4つのメソッドとは、

1. CreateFrame
2. CreateMeshContainer
3. DestroyFrame
4. DestroyMeshContainer

です。分量が多いですが、やっていることはこの4つの仮想関数の実装です。

この実装をここで詳しく解説することは不可能ですし、仕様からくる煩雑性ですので、このコードをそのまま利用するなりしてください。

```
//HRESULT MY_HIERARCHY::CreateFrame(LPCTSTR Name, LPD3DXFRAME *ppNewFrame)
//フレームを作成する
HRESULT MY_HIERARCHY::CreateFrame(LPCTSTR Name, LPD3DXFRAME *ppNewFrame)
{
    HRESULT hr = S_OK;
    MYFRAME *pFrame;
    *ppNewFrame = NULL;

    pFrame = new MYFRAME;
    if (pFrame == NULL)
    {
        return E_OUTOFMEMORY;
    }
    pFrame->Name=new TCHAR[lstrlen(Name) + 1];
    if (!pFrame->Name)
    {
        return E_FAIL;
    }
    strcpy(pFrame->Name,Name);

    D3DXMatrixIdentity(&pFrame->TransformationMatrix);
    D3DXMatrixIdentity(&pFrame->CombinedTransformationMatrix);
    pFrame->pMeshContainer = NULL;
    pFrame->pFrameSibling = NULL;
    pFrame->pFrameFirstChild = NULL;
    *ppNewFrame = pFrame;
    pFrame = NULL;
```

```
        delete pFrame;

        return S_OK;
    }
    //
    //HRESULT MY_HIERARCHY::CreateMeshContainer
    //メッシュコンテナーを作成する
    HRESULT MY_HIERARCHY::CreateMeshContainer(LPCSTR Name, CONST D3DXMESHDATA＊
    pMeshData,
                                       CONST D3DXMATERIAL＊ pMaterials, CONST
    D3DXEFFECTINSTANCE＊ pEffectInstances,
                                              DWORD NumMaterials, CONST DWORD ＊
    pAdjacency, LPD3DXSKININFO pSkinInfo,
                                  LPD3DXMESHCONTAINER ＊ppMeshContainer)
    {

        HRESULT hr;
        MYMESHCONTAINER ＊pMeshContainer = NULL;
        INT iFacesAmount;
        INT iMaterial;
        LPDIRECT3DDEVICE9 pDevice = NULL;
        LPD3DXMESH pMesh = NULL;
        ＊ppMeshContainer = NULL;

        pMesh = pMeshData->pMesh;
        pMeshContainer = new MYMESHCONTAINER;
        if (pMeshContainer == NULL)
        {
            return E_OUTOFMEMORY;
        }
        ZeroMemory(pMeshContainer, sizeof(MYMESHCONTAINER));

        pMeshContainer->Name=new TCHAR[lstrlen(Name) + 1];
        if (!pMeshContainer->Name)
        {
                return E_FAIL;
        }
        strcpy(pMeshContainer->Name,Name);
        pMesh->GetDevice(&pDevice);
        iFacesAmount = pMesh->GetNumFaces();

        // 当該メッシュが法線を持たない場合は法線を追加する
        if (!(pMesh->GetFVF() & D3DFVF_NORMAL))
        {
```

```
        pMeshContainer->MeshData.Type = D3DXMESHTYPE_MESH;
        hr = pMesh->CloneMeshFVF( pMesh->GetOptions(),
                                      pMesh->GetFVF() | D3DFVF_NORMAL,
                                      pDevice, &pMeshContainer->MeshData.pMesh
);
        if (FAILED(hr))
           {
                 return E_FAIL;
           }

        pMesh = pMeshContainer->MeshData.pMesh;
        D3DXComputeNormals( pMesh, NULL );
    }
    else
    {
        pMeshContainer->MeshData.pMesh = pMesh;
        pMeshContainer->MeshData.Type = D3DXMESHTYPE_MESH;
        pMesh->AddRef();
    }
    pMeshContainer->NumMaterials = max(1, NumMaterials);
    pMeshContainer->pMaterials = new D3DXMATERIAL[pMeshContainer-
>NumMaterials];
    pMeshContainer->ppTextures = new LPDIRECT3DTEXTURE9[pMeshContainer-
>NumMaterials];
    pMeshContainer->pAdjacency = new DWORD[iFacesAmount＊3];
    if ((pMeshContainer->pAdjacency == NULL) || (pMeshContainer->pMaterials
== NULL))
    {
        return E_OUTOFMEMORY;
    }

    memcpy(pMeshContainer->pAdjacency, pAdjacency, sizeof(DWORD) ＊
iFacesAmount＊3);
    memset(pMeshContainer->ppTextures, 0, sizeof(LPDIRECT3DTEXTURE9) ＊
pMeshContainer->NumMaterials);

    if (NumMaterials > 0)
    {
        memcpy(pMeshContainer->pMaterials, pMaterials, sizeof(D3DXMATERIAL)
＊ NumMaterials);

        for (iMaterial = 0; iMaterial < NumMaterials; iMaterial++)
        {
             if (pMeshContainer->pMaterials[iMaterial].pTextureFilename !=
NULL)
            {
                TCHAR strTexturePath[MAX_PATH];
                    strcpy( strTexturePath, pMeshContainer-
>pMaterials[iMaterial].pTextureFilename );
                    if( FAILED( D3DXCreateTextureFromFile( pDevice,
strTexturePath,
                                                            &pMeshContainer-
>ppTextures[iMaterial] ) ) )
                    pMeshContainer->ppTextures[iMaterial] = NULL;
                pMeshContainer->pMaterials[iMaterial].pTextureFilename =
NULL;
            }
        }
    }
    else
    {
        pMeshContainer->pMaterials[0].pTextureFilename = NULL;
          memset(&pMeshContainer->pMaterials[0].MatD3D, 0,
sizeof(D3DMATERIAL9));
        pMeshContainer->pMaterials[0].MatD3D.Diffuse.r = 0.5f;
        pMeshContainer->pMaterials[0].MatD3D.Diffuse.g = 0.5f;
        pMeshContainer->pMaterials[0].MatD3D.Diffuse.b = 0.5f;
        pMeshContainer->pMaterials[0].MatD3D.Specular = pMeshContainer-
>pMaterials[0].MatD3D.Diffuse;
    }
    *ppMeshContainer = pMeshContainer;
    pMeshContainer = NULL;

    if (pMeshContainer != NULL)
    {
        DestroyMeshContainer(pMeshContainer);
    }

    return S_OK;
}
//
//HRESULT MY_HIERARCHY::DestroyFrame(LPD3DXFRAME pFrameToFree)
//フレームを破棄する
HRESULT MY_HIERARCHY::DestroyFrame(LPD3DXFRAME pFrameToFree)
{
    SAFE_DELETE_ARRAY( pFrameToFree->Name );
    SAFE_DELETE( pFrameToFree );
    return S_OK;
```

```
}
//
//HRESULT MY_HIERARCHY::DestroyMeshContainer(LPD3DXMESHCONTAINER
pMeshContainerBase)
//メッシュコンテナーを破棄する
HRESULT MY_HIERARCHY::DestroyMeshContainer(LPD3DXMESHCONTAINER
pMeshContainerBase)
{
    INT iMaterial;
    MYMESHCONTAINER ＊pMeshContainer = (MYMESHCONTAINER＊)pMeshContainerBase;

    SAFE_DELETE_ARRAY( pMeshContainer->Name );
    SAFE_DELETE_ARRAY( pMeshContainer->pAdjacency );
    SAFE_DELETE_ARRAY( pMeshContainer->pMaterials );

    if (pMeshContainer->ppTextures != NULL)
    {
        for (iMaterial = 0; iMaterial < pMeshContainer->NumMaterials;
iMaterial++)
        {
            SAFE_RELEASE( pMeshContainer->ppTextures[iMaterial] );
        }
    }

    SAFE_DELETE_ARRAY( pMeshContainer->ppTextures );
    SAFE_RELEASE( pMeshContainer->MeshData.pMesh );

    SAFE_DELETE( pMeshContainer );
    return S_OK;
}
```

One Point

アプリケーション側で実装するような設計にしているのは、読み込みに柔軟性を持たせようという意図だと思います。そのため、このようにかなりのコードを書かなくてはなりませんが…。　アニメーションメッシュの基本についての詳細は、工学社発行の拙書『はじめての3Dゲーム開発』で解説しています。

105 アニメーションメッシュをレンダリングする

再帰コードで、各パーツメッシュをレンダリングする

上級

サンプルプロジェクト　105 アニメーションメッシュ

アニメーションであるが故に、レンダリングには階層の展開が伴います。

アニメーションメッシュのレンダリングには、最低でも次のような関数を作成する必要があります。

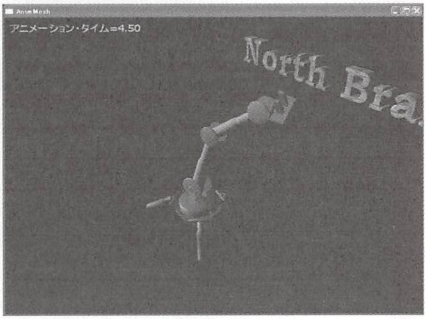
RenderMeshContainer　各メッシュコンテナーをレンダリングする関数

DrawFrame　各フレームをレンダリングする関数

UpdateFrameMatrices　各フレームのワールド変換を行う関数

3つの関数は全て再帰構造になります。

▼実行結果



アニメーション・タイム=4.50

```
//VOID RenderMeshContainer(LPD3DXMESHCONTAINER pMeshContainerBase,
LPD3DXFRAME pFrameBase)
//フレーム内のそれぞれのメッシュをレンダリングする
VOID RenderMeshContainer(LPD3DXMESHCONTAINER pMeshContainerBase, LPD3DXFRAME
pFrameBase)
{
    MYMESHCONTAINER *pMeshContainer = (MYMESHCONTAINER*)pMeshContainerBase;
    MYFRAME *pFrame = (MYFRAME*)pFrameBase;
    UINT iMaterial;

    pDevice->SetTransform(D3DTS_WORLD, &pFrame-
>CombinedTransformationMatrix);

    for (iMaterial = 0; iMaterial < pMeshContainer->NumMaterials;
iMaterial++)
    {
        pDevice->SetMaterial( &pMeshContainer->pMaterials[iMaterial].MatD3D
);
        pDevice->SetTexture( 0, pMeshContainer->ppTextures[iMaterial] );
        pMeshContainer->MeshData.pMesh->DrawSubset(iMaterial);
    }
}
```

```
//VOID DrawFrame(LPD3DXFRAME pFrame)
//複数のメッシュから成るフレームをレンダリングする。
VOID DrawFrame(LPD3DXFRAME pFrame)
{

    LPD3DXMESHCONTAINER pMeshContainer;

    pMeshContainer = pFrame->pMeshContainer;
    while (pMeshContainer != NULL)
    {
        RenderMeshContainer(pMeshContainer, pFrame);

        pMeshContainer = pMeshContainer->pNextMeshContainer;
    }

    if (pFrame->pFrameSibling != NULL)
    {
        DrawFrame(pFrame->pFrameSibling);
    }

    if (pFrame->pFrameFirstChild != NULL)
    {
        DrawFrame(pFrame->pFrameFirstChild);
    }
}
```

```
//VOID UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX
pParentMatrix)
//フレーム内のメッシュ毎にワールド変換行列を更新する
VOID UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX pParentMatrix)
{
    MYFRAME ＊pFrame = (MYFRAME＊)pFrameBase;

    if (pParentMatrix != NULL)
    {
        D3DXMatrixMultiply(&pFrame->CombinedTransformationMatrix, &pFrame-
>TransformationMatrix, pParentMatrix);
    }
    else
    {
        pFrame->CombinedTransformationMatrix = pFrame->TransformationMatrix;
    }

    if (pFrame->pFrameSibling != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameSibling, pParentMatrix);
    }

    if (pFrame->pFrameFirstChild != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameFirstChild, &pFrame-
>CombinedTransformationMatrix);
    }
}
```

ここまで書いて、ようやく次のようにレンダリングできます。

```
// レンダリング
UpdateFrameMatrices(pThing->pFrameRoot, &matWorld);
DrawFrame(pThing->pFrameRoot);
```

One Point

アニメーションメッシュの基本についての詳細は、工学社発行の拙書『はじめての3Dゲーム開発』で解説しています。

106 アニメーションスピードを調整する

●ID3DXAnimationController:: AdvanceTimeメソッドを使う　中級

参考サンプルプロジェクト　105 アニメーションメッシュ

アニメーションの再生スピードは、ID3DXAnimationController::AdvanceTimeメソッドで調節します。

サンプルでは次のようにしています。

```
#define ANIM_SPEED 0.01
     pThing->pAnimController->AdvanceTime(ANIM_SPEED,NULL);
```

メソッド名からも分るように、指定する値は時間の増分です。増分なので、再生スピードを意味します。

単位は秒です。したがって、サンプルでは0.01秒毎フレームです。小さな値ですが、1フレームの増分なので、出力は自然になります。

One Point

第2引数は通常NULLにします。

107 時間指定によりモーションを切り替える

アニメーションタイムを区切って、モーションを分割する

中級

サンプルプロジェクト　107　時間分割によるモーション分け

アニメーションタイムを明示的に指定することにより、モーションを分けることは簡単です。

ID3DXAnimationController::SetTrackPositionメソッドにより、開始タイムを与えてやれば、そこからのモーションに切り替わります。

ただ、そこに指定する時間とモーションの関係を事前に知っていることが必要になります。

Xファイルを作成するCGソフトによってフレームと時間の関係はまちまちです。MAYAではデフォルトでフレーム数が秒数に対応しますが、Lightwave3Dではフレーム数を48倍しないと現実の秒数に対応しないなどという現象も起こったりします。使用するツールによってはXファイルに作成する段階で調整する必要があります。

ここで使用しているXファイルはLightwave3Dで作成したものですが、秒数と上手く対応するように作成しているので、SetTrackPositionには秒数で指定することができます。

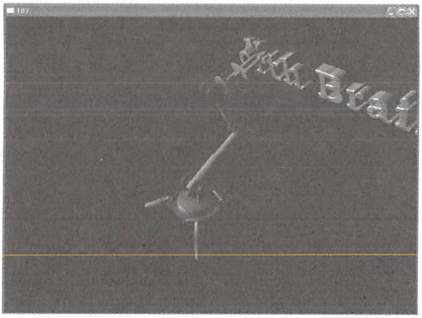
このアニメーションは、全部で12秒あります。

0～6秒までは、ロゴを持ち上げるモーション0、6秒から12秒までは、Nの文字を運ぶモーション1になっています。

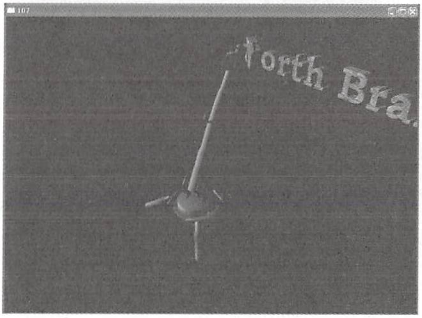
モーション0の開始タイムは0なので、0を指定し、SetTrackPositionを実行します。

モーション1の開始タイムは6なので、6を指定し、SetTrackPositionを実行するだけです。

▼モーション0



▼モーション1



次のように、ウィンドウプロシージャー内で、F1キー、F2キーによりモーションを切り替えるようにしました。

```
        case WM_KEYDOWN:
        switch((CHAR)wParam)
        {
                case VK_ESCAPE:
                        PostQuitMessage(0);
                break;
                case VK_F1://モーション0
                        dwMotion=0;
                        Thing[0].pAnimController->SetTrackPosition(0,0);

                        break;
                case VK_F2://モーション1
                        dwMotion=1;
                        Thing[0].pAnimController->SetTrackPosition(0,6);
                        break;
                case VK_SPACE:
                        boPlayAnim==FALSE ? boPlayAnim=TRUE :
boPlayAnim=FALSE;
                        break;
        }
```

その時のモーションにおけるアニメーション終了地点に到達したら、その時のモーションの開始タイムに戻す必要があります。そうしないと次のモーションに入ってしまうからです。

```
        if(boPlayAnim)
        {
                //アニメーションタイムを進行させる
                fAnimTime+=ANIM_SPEED;
                pThing->pAnimController->AdvanceTime(ANIM_SPEED,NULL);

                //それぞれのモーションにおける終了タイムに達したら、それぞれのモーションの開始タ
イムに戻す
                D3DXTRACK_DESC TrackDesc;
                pThing->pAnimController->GetTrackDesc(0,&TrackDesc);

                if(dwMotion ==0)//モーション0
                {
                        if(TrackDesc.Position>6)
                        {
                                Thing[0].pAnimController->SetTrackPosition(0,0);
                        }
        }
        else if(dwMotion==1)//モーション1
        {
            if(TrackDesc.Position>12)
            {
                Thing[0].pAnimController->SetTrackPosition(0,6);
            }
        }
    }
```

One Point

Xファイルを作成するツール（3DCGソフト）でのフレーム数が実際にどのような時間になるかという特性はおさえておきましょう。

108 トラック切り替えによりモーションを切り替える

●アニメーションコントローラーインターフェイスのメソッドを使う　中級

サンプルプロジェクト　108　トラック切り替えによるモーション分け

Direct3D9のアニメーションには、“トラック”という概念があります。トラックそれぞれに異なるモーションを格納でき、それらのトラックを同時に再生することによりモーションをミックスすることができます。

ここではミックスすることが目的ではありませんが、本来トラックの主たる目的はミックスです。

このTipsを理解するには、Xファイル中の、どの要素がトラックに対応するのかを知る必要があります。

次のリストは単純なアニメーションメッシュのXファイル内部です。

```
xof 0303txt 0032

Material Board_Board {
 0.784314;0.784314;0.784314;1.000000;;
 51.200001;
 0.000000;0.000000;0.000000;;
 0.000000;0.000000;0.000000;;
}
```

```
Frame Frame_SCENE_ROOT {

 FrameTransformMatrix {
  1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,
0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.000000,1.000000;;
 }

 Frame Frame1_Board_Layer1 {

  FrameTransformMatrix {
   1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,
0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.000000,1.000000;;
  }

  Mesh Mesh_Board {
   4;
   -1.550000;-1.500000;0.000000;,
   -1.550000;1.150000;0.000000;,
   2.000000;1.150000;0.000000;,
   2.000000;-1.500000;0.000000;;
   1;
   4;0,1,2,3;;

   MeshMaterialList {
    1;
    1;
    0;
    { Board_Board }
   }
  }
 }
}

AnimationSet AnimationSet_MotionA {

 Animation Animation0 {
  { Frame1_Board_Layer1 }
  AnimationKey {
   0;
   2;
   0;4;1.000000,0.000000,0.000000,0.000000;;,
   60;4;-0.098451,0.000000,-0.995142,0.000000;;;
  }
  AnimationKey {
   1;
   2;
   0;3;1.000000,1.000000,1.000000;;,
   60;3;1.000000,1.000000,1.000000;;;
  }
  AnimationKey {
   2;
   2;
   0;3;0.000000,0.000000,0.000000;;,
   60;3;0.000000,0.000000,0.000000;;;
  }
 }
}
```

トラックに対応するのはAnimationSetです。

このXファイルは今、AnimationSet_MotionA　というトラックを1つ持っています。このXファイルに別のトラックを追加してみましょう。

Xファイル作成ツールが、複数のAnimationSetを出力できないのであれば、手動で作成することができます。どういうことかと言うと、単にAnimationSetの部分を別のXファイルからコピーすればいいのです。

もちろん、別のXファイルは同じジオメトリのアニメーションである必要があります。そうでなければ頂点数が合わず、上手くいきません。

```
xof 0303txt 0032

(中略)

AnimationSet AnimationSet_MotionA {

 Animation Animation0 {
  { Frame1_Board_Layer1 }
  AnimationKey {
   0;
   2;
   0;4;1.000000,0.000000,0.000000,0.000000;;,
   60;4;-0.098451,0.000000,-0.995142,0.000000;;;
  }
  AnimationKey {
   1;
   2;
   0;3;1.000000,1.000000,1.000000;;,
   60;3;1.000000,1.000000,1.000000;;;
  }
  AnimationKey {
   2;
   2;
   0;3;0.000000,0.000000,0.000000;;,
   60;3;0.000000,0.000000,0.000000;;;
  }
 }
}

AnimationSet AnimationSet_MotionB {

 Animation Animation0 {
  { Frame1_Board_Layer1 }
  AnimationKey {
   0;
   2;
   0;4;1.000000,0.000000,0.000000,0.000000;;,
   60;4;-0.064532,0.997916,0.000000,0.000000;;;
  }
  AnimationKey {
   1;
   2;
   0;3;1.000000,1.000000,1.000000;;,
   60;3;1.000000,1.000000,1.000000;;;
  }
  AnimationKey {
   2;
   2;
   0;3;0.000000,0.000000,0.000000;;,
   60;3;0.000000,0.000000,0.000000;;;
  }
 }
}
```

これで、このXファイルを読み込めば、Direct3Dは自動的に2つのトラック（AnimationSet_MotionAとAnimationSet_MotionB）を生成します。

さて、次は実行時にトラックを切り替える方法について考えましょう。

まず初期化時（Xファイル読み込み直後）に、アニメーションコントローラーからアニメーションセットを取得しておきます。サンプルではグローバルに用意した、アニメーションセットインターフェイスの配列に格納しています。

このアニメーションセットインターフェイスpAnimSetはトラック切り替え時に使用します。

```
//アニメーショントラックを得る
for(DWORD i=0;i<pThing->pAnimController->GetNumAnimationSets();i++)
{
   pThing->pAnimController->GetAnimationSet(i,&pAnimSet[i]);
}
```

ウィンドウプロシージャー内でF1キーとF2キーにより、トラックを切り替えます。

まず、D3DXTRACK_DESC構造体を用意します。

```
D3DXTRACK_DESC TrackDesc;
```

それぞれのキーが押されたら、D3DXTRACK_DESC構造体に意図する値を代入して、ID3DXAnimationController::SetTrackDescメソッドを実行します。

次に、ID3DXAnimationController::SetTrackAnimationSetメソッドにより、保存しておいてそれぞれのアニメーションセットをセットします。

最後にID3DXAnimationController::SetTrackEnableにより、無効にしたいトラックをFALSEに、有効にしたいトラックをTRUEに設定します。

```
        case WM_KEYDOWN:
        switch((CHAR)wParam)
        {
             case VK_ESCAPE:
                  PostQuitMessage(0);
             break;
             case VK_F1:
                  TrackDesc.Weight=1;
                  TrackDesc.Speed=1;
                  TrackDesc.Enable=1;
                  Thing[0].pAnimController->SetTrackDesc(0,&TrackDesc);
                  Thing[0].pAnimController->SetTrackAnimationSet
(0,pAnimSet[0]);
                  Thing[0].pAnimController->SetTrackEnable(0,TRUE);
                  Thing[0].pAnimController->SetTrackEnable(1,FALSE);
                  break;
             case VK_F2:
                  TrackDesc.Weight=1;
                  TrackDesc.Speed=1;
                  TrackDesc.Enable=1;
                  Thing[0].pAnimController->SetTrackDesc(1,&TrackDesc);
                  Thing[0].pAnimController->SetTrackAnimationSet
(1,pAnimSet[1]);
```

```
                        Thing[0].pAnimController->SetTrackEnable(0,FALSE);
                        Thing[0].pAnimController->SetTrackEnable(1,TRUE);
                        break;
                case VK_SPACE:
                        boPlayAnim==FALSE ? boPlayAnim=TRUE :
boPlayAnim=FALSE;
                        break;
          }
```

なお、初期化時に、

```
TrackDesc.Weight=1;
TrackDesc.Speed=1;
TrackDesc.Enable=1;
Thing[0].pAnimController->SetTrackDesc(1,&TrackDesc);
Thing[0].pAnimController->SetTrackAnimationSet(1,pAnimSet[1]);
```

までを行っておくこともできます。その場合、
実行時での切り替えは

```
case VK_F1:
    Thing[0].pAnimController->SetTrackEnable(0,TRUE);
    Thing[0].pAnimController->SetTrackEnable(1,FALSE);
    break;
case VK_F2:
    Thing[0].pAnimController->SetTrackEnable(0,FALSE);
    Thing[0].pAnimController->SetTrackEnable(1,TRUE);
    break;
```

と書くだけで済みます。

One Point

トラックをミックス（モーションをミックス）したい場合は、両方のトラックを有効（SetTrackEnablewoをTRUE）にして、TrackDesc のWeightメンバを0.5などとします。

109 スキンメッシュとは

●メッシュを構成する頂点そのものを動かすメッシュ

初級

スキンメッシュとは、メッシュを構成する頂点そのものを動かすメッシュです。頂点が動けばメッシュとしての形状が時間軸で変化することになるので、単体でアニメーションすることができます。

通常のアニメーションメッシュにおいて、個々のパーツメッシュ自体は頂点の位置がローカル系で固定されているので、言ってみれば“剛体”です。それに対し、スキンメッシュは頂点のローカル位置が変化するので、特に有機体や流体の表現に適しています。

また、通常のメッシュアニメーションであれば、どうしても個々のパーツ相互に繋ぎ目が見えてしまいますが、スキンメッシュのモデリングは文字通りワンスキンにすることができます。人体モデルの滑らかなアニメーションにおいても、間接部の繋ぎ目を無くすことができるので、通常のフレームアニメーションより適しています。

例えば顔メッシュにおいて、表情の変化を通常のメッシュで表現するのは極めて難しく、現実的に不可能とも言えます。

One Point

スキンメッシュの基本についての詳細は、工学社発行の拙書『ゲームコーディングvol.1　Direct3D・COM編』で解説しています。

110 スキンメッシュの作成について

●3DCGソフトでの作成過程を見る

中級

Tips074「メッシュの作成について」でも述べましたが、ここで言う“作成”とは、外部ファイル等による静的な作成のことです。つまり素材としての作成です。素材作成の段階はDirectXの範囲外ですが、けっこう重要なことだと思うので記しておきます。

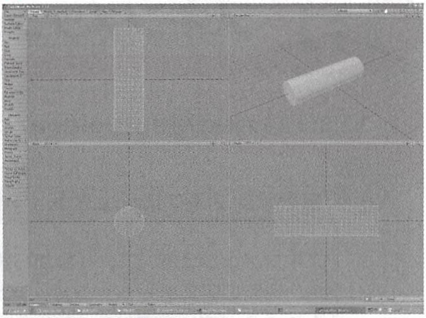
アニメーションメッシュとなると、もはやコード中での動的な作成は現実的ではありませんし、そのような素材生成関数も用意されていません（もちろんアニメーションメッシュを読み込んだり、操作する関数はあります）。作成は通常のメッシュと同じく、3DCGソフトを使用します。

ここでは、筆者が使用しているLightwave3Dという3DCGソフトを例に解説します。

右の図はLightWaveでモデリングし、Xファイルに出力する過程です。

全体のアニメーションオブジェクトを構成する各パーツオブジェクトの作成は、通常の単一メッシュと同じです。異なるのは、“アニメーション付け（モーション付け）”という工程が加わる部分です。

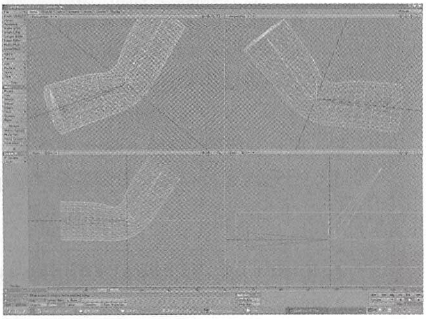
まず、メッシュをモデリングします。本サンプルでは単純なシリンダーをモデルとして使用します。



▼ワンスキンとしてモデリング

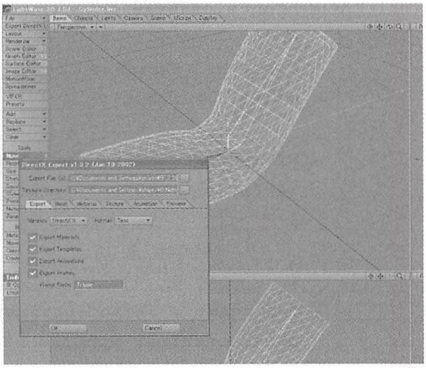
次に、動き（モーション）を付けます。スキンメッシュとする場合、たった一つのオブジェクトでモーションが実現できます。

メッシュの変形は、“ボーン”を仕込み、ボーンを動かすことにより行います。



▼「ボーン」で変形することにより、1つのオブジェクトで動きを表現する

ここまでできれば、あとは通常のメッシュの場合と同じように、Xファイルコンバーターでエクスポートします。



▼Xファイルとして出力

DirectX Export v1.3.2 (Jan 19 2002)
Export
Mesh
Material
Texture
Animation
Preview
Export Materials
Export Templates
Export Animations
Export Frames
OK
Cancel

One Point

3DCGソフトの購入を考えている場合は、Tips028とコラムが多少なりとも参考になると思います。

111 スキンメッシュをXファイルから読み込む

D3DXLoadMeshHierarchyFromX関数にを使う

中級

参考サンプルプロジェクト　112 スキンメッシュのレンダリング

スキンメッシュをXファイルから読み込む場合、通常のアニメーションメッシュの読み込みにくらべて、処理が多くなります。

アニメーションメッシュ全般において、

1. ID3DXAllocateHierarchy抽象クラスを派生したクラスを定義し、必須である4つのメソッドを実装する。

という処理は必ず必要になります。

スキンメッシュの場合は、それに加えて、

2. ボーン行列関連を初期化する。

という処理を行わなくてはなりません。

1. についてはTips104で解説しています。2. の処理は、サンプルSKIN_MESHクラス内のAllocateAllBoneMatricesメソッド及び、そのサブ関数であるAllocateBoneMatrixメソッドで行っています。

ボーンの初期化コードを一応掲載しますが、この実装をここで詳しく解説することは不可能ですし、仕様からくる煩雑性ですので、このコードをそのまま利用するなりしてください。

```
//HRESULT AllocateBoneMatrix( THING* pThing,LPD3DXMESHCONTAINER
pMeshContainerBase )
//
HRESULT SKIN_MESH::AllocateBoneMatrix( THING* pThing,LPD3DXMESHCONTAINER
pMeshContainerBase )
{
    MYFRAME *pFrame=NULL;
    DWORD dwBoneAmt=0;

    MYMESHCONTAINER *pMeshContainer = (MYMESHCONTAINER*)pMeshContainerBase;
    if (pMeshContainer->pSkinInfo == NULL)
    {
        return S_OK;
    }
    dwBoneAmt = pMeshContainer->pSkinInfo->GetNumBones();
    pMeshContainer->ppBoneMatrix = new D3DXMATRIX*[dwBoneAmt];

    for (DWORD i=0; i<dwBoneAmt; i++)
    {
        pFrame = (MYFRAME*)D3DXFrameFind( pThing->pFrameRoot,
pMeshContainer->pSkinInfo->GetBoneName(i) );
        if (pFrame == NULL)
```

```
          {
                return E_FAIL;
          }
          pMeshContainer->ppBoneMatrix[i] = &pFrame->CombinedTransformationMatrix;

    }
    return S_OK;
}
```

```
//HRESULT AllocateAllBoneMatrices( THING＊ pThing,LPD3DXFRAME pFrame )
//
HRESULT SKIN_MESH::AllocateAllBoneMatrices( THING＊ pThing,LPD3DXFRAME pFrame )
{
    if (pFrame->pMeshContainer != NULL)
    {
        if(FAILED(AllocateBoneMatrix(pThing,pFrame->pMeshContainer)))
          {
                return E_FAIL;
          }
    }
    if (pFrame->pFrameSibling != NULL)
    {
        if(FAILED(AllocateAllBoneMatrices(pThing,pFrame->pFrameSibling)))
          {
                return E_FAIL;
          }
    }
    if (pFrame->pFrameFirstChild != NULL)
    {
        if(FAILED(AllocateAllBoneMatrices(pThing,pFrame->pFrameFirstChild)))
          {
                return E_FAIL;
          }
    }
    return S_OK;
}
```

One Point

スキンメッシュの基本についての詳細は、工学社発行の拙書『ゲームコーディングvol.1　Direct3D・COM編』で解説しています。

112 スキンメッシュをレンダリングする

●再帰コードにより、各ボーンそれぞれについて影響されるポリゴンをレンダリング

上級

サンプルプロジェクト　112 スキンメッシュのレンダリング

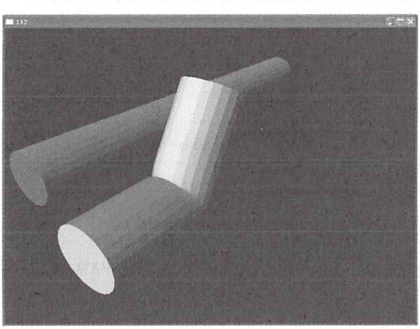
パーツメッシュの階層によるアニメーションでは、階層ノードにあるものはパーツメッシュでした。ところが、スキンメッシュの場合、そもそもパーツメメッシュなどはなく、単一のメッシュ（ワンスキン）です。

では、スキンメッシュの場合は階層云々という処理は不要なのかというと、そうではありません。

スキンメッシュにおいても、通常のアニメーションと同様に、階層の展開・走査は必要になります。ただ、階層ノードにあるものがパーツメッシュではなくボーンに置き換わっただけです。

次にサンプルから、レンダリングに必要な3つの関数を抜粋します。

▼奥にあるのはマルチトラックのファイルから読み込んだもの。トラックを切り替えないので、トラック0(モーションA)のみ再生されている。



```
//VOID RenderMeshContainer(LPDIRECT3DDEVICE9 pDevice,MYMESHCONTAINER＊
pMeshContainer, MYFRAME＊ pFrame)
//フレーム内のそれぞれのメッシュをレンダリングする
VOID SKIN_MESH::RenderMeshContainer(LPDIRECT3DDEVICE9
pDevice,MYMESHCONTAINER＊ pMeshContainer, MYFRAME＊ pFrame)
{
    DWORD i,k,m;
    DWORD dwBoneIndex,dwBlendMatrixAmt;
    DWORD dwPrevBoneID;
    LPD3DXBONECOMBINATION pBoneCombination;
    UINT iMatrixIndex;
    D3DXMATRIX matStack;
    //スキンメッシュの場合
    if(pMeshContainer->pSkinInfo != NULL)
    {
        pBoneCombination = (LPD3DXBONECOMBINATION)pMeshContainer-
>pBoneBuffer->GetBufferPointer();
```

```
        dwPrevBoneID=UINT_MAX;
        for(i = 0; i < pMeshContainer->dwBoneAmt; i++)
        {
            dwBlendMatrixAmt=0;
            for(k = 0; k< pMeshContainer->dwWeight; k++)
            {
                if (pBoneCombination[i].BoneId[k] != UINT_MAX)
                {
                    dwBlendMatrixAmt=k;
                }
            }
            pDevice->SetRenderState(D3DRS_VERTEXBLEND, dwBlendMatrixAmt);
            for(k = 0; k < pMeshContainer->dwWeight; k++)
            {
                iMatrixIndex = pBoneCombination[i].BoneId[k];
                if (iMatrixIndex != UINT_MAX)
                {
                    matStack=pMeshContainer->pBoneOffsetMatrices[iMatrixIndex] * (*pMeshContainer->ppBoneMatrix[iMatrixIndex]);
                    pDevice->SetTransform( D3DTS_WORLDMATRIX(k), &matStack );
                }
            }
            pDevice->SetMaterial( &pMeshContainer->pMaterials[pBoneCombination[i].AttribId].MatD3D );
            pDevice->SetTexture( 0, pMeshContainer->ppTextures[pBoneCombination[i].AttribId] );
            dwPrevBoneID = pBoneCombination[i].AttribId;
            pMeshContainer->MeshData.pMesh->DrawSubset(i);
        }
    }
    //通常メッシュの場合
    else
    {
        pDevice->SetTransform(D3DTS_WORLD, &pFrame->CombinedTransformationMatrix);
        for (i = 0; i < pMeshContainer->NumMaterials; i++)
        {
            pDevice->SetMaterial( &pMeshContainer->pMaterials[i].MatD3D );
            pDevice->SetTexture( 0, pMeshContainer->ppTextures[i] );
            pMeshContainer->MeshData.pMesh->DrawSubset(i);
            }
    }
}
```

```
//VOID DrawFrame(LPDIRECT3DDEVICE9 pDevice,LPD3DXFRAME pFrameBase)
//フレームをレンダリングする。
VOID SKIN_MESH::DrawFrame(LPDIRECT3DDEVICE9 pDevice,LPD3DXFRAME pFrameBase)
{
    MYFRAME＊ pFrame=(MYFRAME＊)pFrameBase;
    MYMESHCONTAINER＊ pMeshContainer = (MYMESHCONTAINER＊)pFrame-
>pMeshContainer;

    while (pMeshContainer != NULL)
    {
        RenderMeshContainer(pDevice,pMeshContainer, pFrame);

        pMeshContainer = (MYMESHCONTAINER＊)pMeshContainer-
>pNextMeshContainer;
    }
    if (pFrame->pFrameSibling != NULL)
    {
        DrawFrame(pDevice,pFrame->pFrameSibling);
    }
    if (pFrame->pFrameFirstChild != NULL)
    {
        DrawFrame(pDevice,pFrame->pFrameFirstChild);
    }
}
```

```
//VOID UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX
pParentMatrix)
//フレーム内のメッシュ毎にワールド変換行列を更新する
VOID SKIN_MESH::UpdateFrameMatrices(LPD3DXFRAME pFrameBase, LPD3DXMATRIX
pParentMatrix)
{
    MYFRAME ＊pFrame = (MYFRAME＊)pFrameBase;

    if (pParentMatrix != NULL)
    {
        D3DXMatrixMultiply(&pFrame->CombinedTransformationMatrix, &pFrame-
>TransformationMatrix, pParentMatrix);
    }
    else
    {
        pFrame->CombinedTransformationMatrix = pFrame->TransformationMatrix;
    }
    if (pFrame->pFrameSibling != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameSibling, pParentMatrix);
    }
    if (pFrame->pFrameFirstChild != NULL)
    {
        UpdateFrameMatrices(pFrame->pFrameFirstChild, &pFrame-
>CombinedTransformationMatrix);
    }
}
```

One Point

スキンメッシュの基本についての詳細は、工学社発行の拙書『ゲームコーディングvol.1 Direct3D・COM編』で解説しています。

113 スキンアニメーションスピードを調整する

●ID3DXAnimationController:: AdvanceTimeメソッドを使う 中級

参考サンプルプロジェクト 112 スキンメッシュのレンダリング

アニメーションの再生スピードは、ID3DXAnimationController::AdvanceTimeメソッドで調節します。サンプルでは次のようにしています。

```
pThing->pAnimController->AdvanceTime(fAnimTime-fAnimTimeHold,NULL);
fAnimTimeHold=fAnimTime;
if(boPlayAnim)
{
    fAnimTime+=0.01f;
}
```

メソッド名からも分るように、指定する値は時間の増分です。増分なので、再生スピードを意味します。単位は秒です。したがって、サンプルでは0.01秒毎フレームです。小さな値ですが、1フレームの増分なので、出力は自然になります。

One Point

アニメーションスピードの調整は、アニメーションメッシュ全般で同じ仕様です。したがって、Tips106と全く同じです（ただ、たまたまここでは書き方を微妙に変えていますが）。

114 時間指定によりモーションを切り替える

●ID3DXAnimationController:: SetTrackPositionメソッドを使う　中級

サンプルプロジェクト　114　時間分割によるモーション分け

アニメーションタイムを明示的に指定することにより、モーションを分けることは簡単です。

ID3DXAnimationController::SetTrackPositionメソッドにより、開始タイムを与えてやれば、そこからのモーションに切り替わります。

ただ、そこに指定する時間とモーションの関係を事前に知っていることが必要になります。

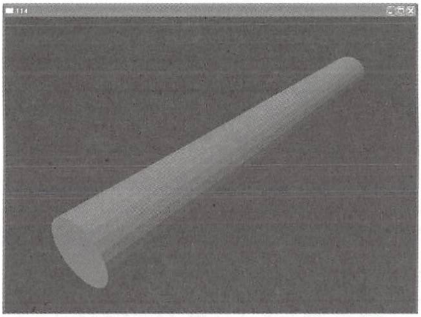
Xファイルを作成するCGソフトによってフレームと時間の関係はまちまちです。MAYAではデフォルトでフレーム数が秒数に対応しますが、Lightwave3Dではフレーム数を48倍しないと現実の秒数に対応しないなどという減少も起こったりします。使用するツールによってはXファイルに作成する段階で調整する必要があります。

ここで使用しているXファイルはLightwave3Dで作成したものですが、秒数と上手く対応するように作成しているので、SetTrackPositionには秒数で指定することができます。

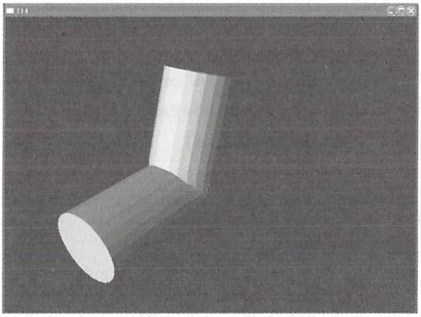
このアニメーションは、全部で1.4秒あります。

0～0.7秒までは、折れ曲がるモーション0、0.7秒から1.4秒までは、伸縮するモーション1になっています。

▼0.7～1.4秒のモーション



▼0.0～0.7秒のモーション



モーション0の開始タイムは0なので、0を指定し、SetTrackPositionを実行します。モーション1の開始タイムは0.7なので、0.7を指定し、SetTrackPositionを実行するだけです。

次のように、ウィンドウプロシージャー内で、F1キー、F2キーによりモーションを切り替えるようにしました。

```
case WM_KEYDOWN:
switch((CHAR)wParam)
{
    case VK_ESCAPE:
        PostQuitMessage(0);
    break;
    case VK_F1://モーション0
        dwMotion=0;
        Thing[0].pAnimController->SetTrackPosition(0,0);

        break;
    case VK_F2://モーション1
        dwMotion=1;
        Thing[0].pAnimController->SetTrackPosition(0,0.7);
        break;
    case VK_SPACE:
        boPlayAnim==FALSE ? boPlayAnim=TRUE : boPlayAnim=FALSE;
        break;
}
```

その時のモーションにおけるアニメーション終了地点に到達したら、その時のモーションの開始タイムに戻す必要があります。そうしないと次のモーションに入ってしまうからです。

```
if(boPlayAnim)
{
     //アニメーションタイムを進行させる
    fAnimTime+=ANIM_SPEED;
    pThing->pAnimController->AdvanceTime(ANIM_SPEED,NULL);

    //それぞれのモーションにおける終了タイムに達したら、それぞれのモーションの開始タイムに戻す
    D3DXTRACK_DESC TrackDesc;
    pThing->pAnimController->GetTrackDesc(0,&TrackDesc);

    if(dwMotion ==0)//モーション0
    {
```

```
                if(TrackDesc.Position>0.7)
                {
                    Thing[0].pAnimController->SetTrackPosition(0,0);
                }
            }
            else if(dwMotion==1)//モーション1
            {
                if(TrackDesc.Position>1.4)
                {
                    Thing[0].pAnimController->SetTrackPosition(0,0.7);
                }
            }
        }
```

One Point

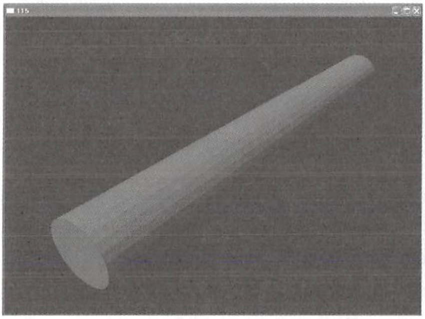
スキンメッシュ固有の処理ではないので、原理はTips107と全く同じです。

115 トラック切り替えによりモーションを切り替える

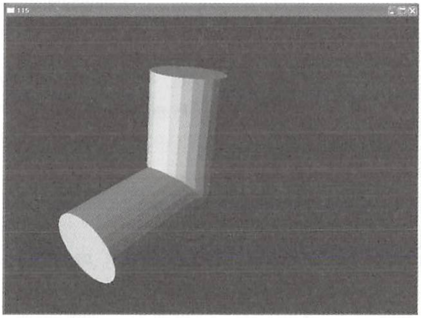
●アニメーションコントローラーインターフェイスのメソッドを使う　中級

サンプルプロジェクト　115　トラック切り替えによるモーション分け

▼実行結果



▼実行結果



ウィンドウプロシージャー内で、キー入力により、トラックを切り替えています。解説については、Tips108と全く同じ議論になりますので、Tips108を参照してください。

```
        case WM_KEYDOWN:
        switch((CHAR)wParam)
        {
            case VK_ESCAPE:
                PostQuitMessage(0);
            break;
            case VK_F1:
                TrackDesc.Weight=1;
                TrackDesc.Speed=1;
                TrackDesc.Enable=1;
                Thing[0].pAnimController->SetTrackDesc(0,&TrackDesc);
                Thing[0].pAnimController-
>SetTrackAnimationSet(0,pAnimSet[0]);
                Thing[0].pAnimController->SetTrackEnable(0,TRUE);
                Thing[0].pAnimController->SetTrackEnable(1,FALSE);
                break;
            case VK_F2:
                TrackDesc.Weight=1;
                TrackDesc.Speed=1;
                TrackDesc.Enable=1;
                Thing[0].pAnimController->SetTrackDesc(1,&TrackDesc);
                Thing[0].pAnimController-
>SetTrackAnimationSet(1,pAnimSet[1]);
                Thing[0].pAnimController->SetTrackEnable(0,FALSE);
                Thing[0].pAnimController->SetTrackEnable(1,TRUE);
                break;
            case VK_SPACE:
                boPlayAnim==FALSE ? boPlayAnim=TRUE :
boPlayAnim=FALSE;
                break;
        }
```

パーティクル

116 ポイントスプライトとは

●ポイントスプライトの意義と特徴を理解する

初級

粒子であるパーティクルの表現は、ポイントスプライト機能を利用すれば便利です。

最近のグラフィックカードには「ポイントスプライト」という機能が実装されていて、Direct3Dからも操作できます（というよりも、Direct3Dを意識してハードウェア実装されています）。

ポイントスプライトはどのようなものなのでしょう？

通常、スプライトをレンダリングするには、四角ポリゴンあるいはID3DXSpriteオブジェクトを使用しますが、パーティクルのように、大量の粒子を表現したい場合、それらでは不便です。

もちろん、それらにより表現することはできますが、もっと使いやすようにと考えられたのが「ポイントスプライト」です。

ポイントスプライトは、ポイント（頂点）ただ1つだけで、テクスチャを貼り付けることができます。つまり、たった1つの頂点で。あたかも四角ポリゴンあるいは通常のスプライトオブジェクトであるかのように振舞ってくれます。頂点1つ1つにテクスチャを貼り付けることができるので、まさにパーティクルの粒子1つと頂点が一対一に結びつき、粒子の表現、及び実装においても格好のパラダイムと言えます。

さて、では実際にどうやってポイントスプライトを実装するでしょう？　ポイントスプライトは通常のスプライトであるID3DXSpriteのようにインターフェイスとして用意されているわけではなく、もっと簡単にレンダリングステートの設定だけで実現できます。

```
pDevice->SetRenderState( D3DRS_POINTSPRITEENABLE ,TRUE );
```

基本的にはこれだけです。あとは当然スプライトなので、テクスチャをセットするコードくらいです。

```
pDevice->SetTexture(0,pTexture);
```

OnePoint

グラフィックカードによっては、ポイントスプライトをサポートしていない場合があり、その場合はDirect3Dによってエミュレートされます。

117 ポイントスプライトをレンダリングする（カメラ空間）

3D空間であるカメラ空間内でポイントスプライトをレンダリングする 中級

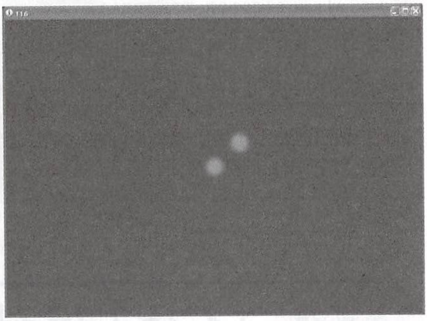
サンプルプロジェクト 117 ポイントスプライト（カメラ空間）

ポイントスプライトは、カメラ空間（3D空間）とスクリーン空間（ディスプレイ平面）においてレンダリングすることができ、それぞれの場合で、位置や大きさの単位が異なりますので、その部分において扱いが異なります。

ここでは、カメラ空間内にポイントスプライトをレンダリングします。

まず、頂点を定義します。頂点タイプは単純なもので、座標と色だけを持つもので十分です。

▼実行結果



```
struct COLOR_VERTEX
{
    D3DXVECTOR3 vecCoord;
    DWORD dwColor;
};
```

その頂点型で、インスタンスを作成します。次のコードは2つの頂点を作成しています。

```
COLOR_VERTEX PointSprite[2]=
{
    D3DXVECTOR3(0,0,0),D3DCOLOR_ARGB(255,255,255,255),
    D3DXVECTOR3(0.1,0.1,0),D3DCOLOR_ARGB(255,255,255,255)
};
```

頂点をポイントスプライトとしてレンダリングするためにすることは、レンダリングステートを次のようにするだけです。

```
pDevice->SetRenderState( D3DRS_POINTSPRITEENABLE ,TRUE );
```

ポイントスプライトに貼り付けたいテクスチャをセットします。

```
pDevice->SetTexture(0,pTexture);
```

頂点の型をセットします。

```
pDevice->SetFVF(D3DFVF_XYZ |  D3DFVF_DIFFUSE );
```

あとはレンダリングするだけです。ここでは、上で初期化した2つの頂点を2つのポイントスプライトとしてレンダリングしています。

```
pDevice->DrawPrimitiveUP(D3DPT_POINTLIST, 2, PointSprite,
sizeof(COLOR_VERTEX));
```

One Point

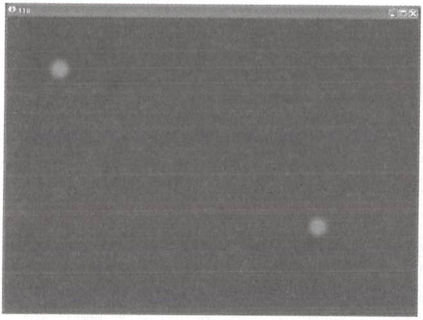
頂点の定義において、座標にD3DXVECTOR3型を使用しましたが、単にFLOAT型3つ（x,y,z3成分）で表現しても全く同じです。

118 ポイントスプライトをレンダリングする(スクリーン空間)

2D平面であるスクリーン空間上にポイントスプライトをレンダリングする　中級

サンプルプロジェクト　118　ポイントスプライト（スクリーン空間）
サンプルプロジェクト　118-2　スクリーン空間、頂点自身でサイズを持つ

▼レンダリングステートでサイズ設定



ポイントスプライトは、カメラ空間（3D空間）とスクリーン空間（ディスプレイ平面）においてレンダリングすることができ、それぞれの場合で、位置や大きさの単位が異なりますので、その部分において扱いが異なります。

ここでは、スクリーン空間上にポイントスプライトをレンダリングします。

まず、頂点を定義します。座標、色を持たせます。スクリーン空間へのレンダリングをするのであれば、座標は投影後のものなので4つの成分が必要です（w成分）。

```
struct COLOR_VERTEX
{
    D3DXVECTOR3 vecCoord;
    FLOAT w;
    DWORD dwColor;
};
```

その型で、インスタンスを作成します。次のコードは2つの頂点を作成しています。

```
COLOR_VERTEX PointSprite[2]=
{
    D3DXVECTOR3(100,100,0),1,D3DCOLOR_ARGB(255,255,255,255),
    D3DXVECTOR3(600,400,0),1,D3DCOLOR_ARGB(255,255,255,255)
};
```

頂点をポイントスプライトとしてレンダリングするためにすることは、レンダリングステートを次のようにするだけです。

```
pDevice->SetRenderState( D3DRS_POINTSPRITEENABLE ,TRUE );
```

ポイントスプライトに貼り付けたいテクスチャをセットします。

```
pDevice->SetTexture(0,pTexture);
```

頂点の型をセットします。

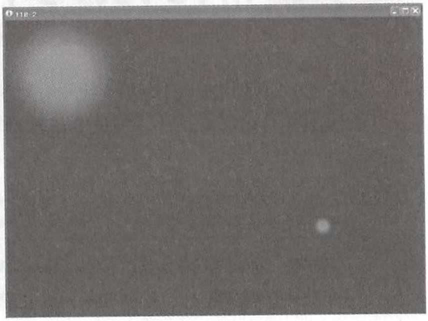
```
pDevice->SetFVF(D3DFVF_XYZW | D3DFVF_DIFFUSE );
```

あとはレンダリングするだけです。ここでは、上で初期化した2つの頂点を2つのポイントスプライトとしてレンダリングしています。

```
pDevice->DrawPrimitiveUP(D3DPT_POINTLIST, 2, PointSprite,
sizeof(COLOR_VERTEX));
```

なお、ここではポイントスプライトのサイズをレンダリングステートで設定していますが、レンダリングステートではなく、頂点自身にサイズ情報を持たせることもできます。サンプル118-2がその方式を採っています。

▼頂点自身にサイズ情報を持たせている場合



その場合、定義は

```
struct COLOR_VERTEX
{
    D3DXVECTOR3 vecCoord;
    FLOAT w;
    FLOAT fScale;
    DWORD dwColor;
};
```

となり、また初期化は

```
COLOR_VERTEX PointSprite[2]=
{
    D3DXVECTOR3(100,100,0),1,300,D3DCOLOR_ARGB(255,255,255,255),
    D3DXVECTOR3(600,400,0),1,50,D3DCOLOR_ARGB(255,255,255,255)
};
```

といったように、サイズ情報分だけ増えます。

そして、頂点の型をパイプラインに設定する場合も、

```
pDevice->SetFVF(D3DFVF_XYZW | D3DFVF_PSIZE | D3DFVF_DIFFUSE );
```

となります（D3DFVF_PSIZEが増える）。

他の部分は同様です。

One Point

頂点の定義において、座標にD3DXVECTOR3型を使用しましたが、単にFLOAT型4つ（x,y,z,wの4成分）で表現しても全く同じです。

119 ポイントスプライトのサイズを変更する(カメラ空間)

●IDirect3DDevice9::SetRenderState(D3DRS_POINTSIZE,XXX) を設定することでサイズ変更を反映させる

中級

サンプルプロジェクト　119　サイズを変更する（カメラ空間）

まず、ウィンドウプロシージャー内で、サイズ保持用変数としてグローバルに宣言しているfScaleをキー入力により増減させます。

```
case WM_KEYDOWN:
    switch(wParam)
    {
    case VK_LEFT:
            fScale-=(FLOAT)(fScale>0)＊0.01;
            break;
    case VK_RIGHT:
            fScale+=(FLOAT)(fScale<1.0)＊0.01;
            break;
    }
    break;
```

留意しなくてはならないことは、値が0.0から1.0の範囲で十分ということです。カメラ空間（3D空間）の場合はこのようになります。スクリーン空間の場合は、ピクセル単位になるのでもっと大きな値で増減させます。

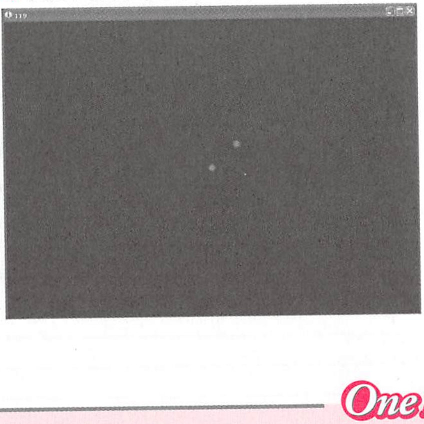
ポイントスプライトのサイズを変更できるようにするためには、次のステートを設定します。

```
pDevice->SetRenderState( D3DRS_POINTSCALEENABLE,  TRUE );
```

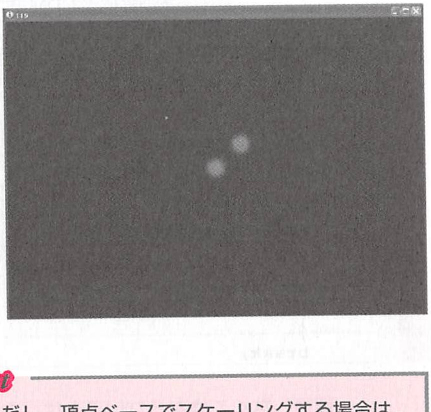
実際にサイズを変更するには、次のレンダリングステートで変更します。

```
pDevice->SetRenderState( D3DRS_POINTSIZE,GET_ADDRESS(fScale) );
```

▼サイズ=0.01（16ピクセル）



▼サイズ=1.0(64ピクセル)（筆者のQuadroFXでも64が上限だった）



One Point

多くのグラフィックカードにおいて、パフォーマンス上の理由から、ポイントスプライトは最終的に64ピクセルに制限されています。その場合fScaleを1.0以上にしても最終的なピクセルサイズは64ピクセル以上にはなりません。

ただし、頂点ベースでスケーリングする場合は、いくらでも大きくレンダリングすることが可能です。その方法については120-2サンプルあるいは118-2サンプルを参照してください。

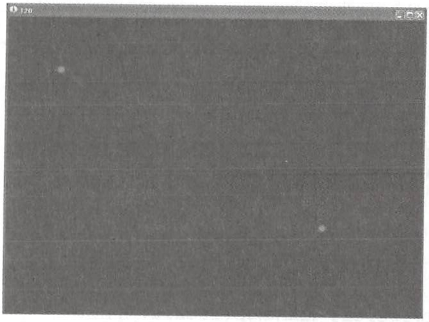
120 ポイントスプライトのサイズを変更する(スクリーン空間)

IDirect3DDevice9::SetRenderState(D3DRS_POINTSIZE,XXX)を設定することでサイズ変更を反映させ、同時に頂点自身にもサイズ情報を付加する

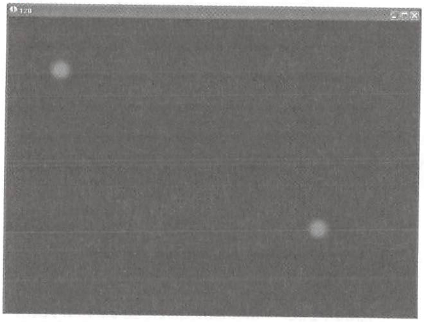
中級

サンプルプロジェクト　120　サイズを変更する（スクリーン空間）
サンプルプロジェクト　120-2　サイズを変更する(スクリーン空間、頂点ベース)

▼16ピクセル



▼64ピクセル



まず、ウィンドウプロシージャー内で、サイズ保持用変数としてグローバルに宣言しているfSizeをキー入力により増減させます。

```
case WM_KEYDOWN:
    switch(wParam)
    {
    case VK_LEFT:
            fSize-=fSize>0;
            break;
    case VK_RIGHT:
            fSize+=fSize<800;
            break;
    }
    break;
```

留意しなくてはならないことは、カメラ空間（3D空間）での時と異なり、値が0から800という大きな範囲であるということですスクリーン空間の場合は、ピクセル単位になるからです。ただ、800というのは実は大きすぎで、実際は64ピクセルあたりが多くのグラフィックカードでの上限です。

ポイントスプライトのサイズを変更できるようにするためには、次のステートを設定します。

```
pDevice->SetRenderState( D3DRS_POINTSCALEENABLE,  TRUE );
```

実際にサイズを変更するには、次のレンダリングステートで変更します。

```
pDevice->SetRenderState( D3DRS_POINTSIZE,GET_ADDRESS(fSize) );
```

なお、サイズの変更は、このようなレンダリングステートでの変更のほかに、頂点自身にサイズ情報を持たせ、頂点そのものをスケーリングすることもできます。サンプル120-2はその手法を採っています。

この手法ではまず頂点宣言が異なります。

```
struct COLOR_VERTEX
{
    D3DXVECTOR3 vecCoord;
    FLOAT w;
    FLOAT fSize;
    DWORD dwColor;
};
```

サイズ情報のメンバが増えていることが分かるでしょう。

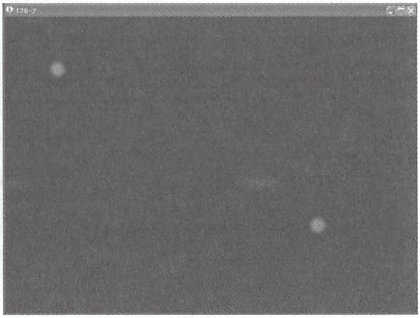
そして、頂点のインスタンスを作成する時も当然サイズを指定します。ここでは、固定値ではなくfSizeを渡すことにより、その時点でのサイズで動的にサイズを変換させています。

```
COLOR_VERTEX PointSprite[2]=
{
    D3DXVECTOR3(100,100,0),1,fSize,D3DCOLOR_ARGB(255,255,255,255),
    D3DXVECTOR3(600,400,0),1,fSize,D3DCOLOR_ARGB(255,255,255,255)
};
```

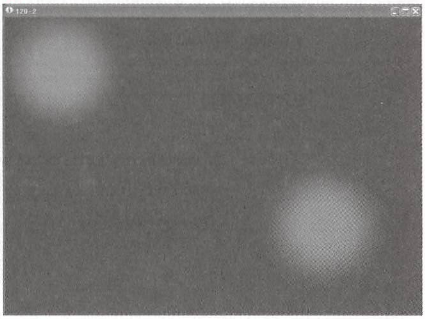
この手法の場合、D3DRS_POINTSIZEステートの設定は不要です。というよりも、設定しても無視されるので意味がありません。

頂点自体にサイズを持たせた場合は、かなり大きくレンダリングすることが可能になります。

▼64ピクセル



▼240ピクセル



One Point

多くのグラフィックカードにおいて、パフォーマンス上の理由から、ポイントスプライトは最終的に64ピクセルに制限されています。その場合fScaleを64以上にしても最終的なピクセルサイズは64ピクセル以上にはなりません。

このサンプルで800ピクセルまで考慮しているのは、将来の高性能なカードのためであって、800ピクセルまで拡大できるわけではありません。

121 ポイントスプライトをパーティクルらしくレンダリングする

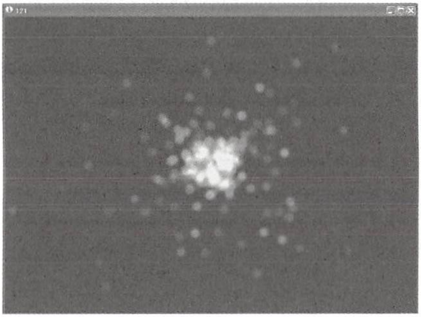
●アルファブレンディング設定をして見栄えを良くする

中級

サンプルプロジェクト　121　パーティクルらしく

まず、ポイントスプライト用の頂点を定義します。

▼実行結果



```
struct PRTICLE
{
     D3DXVECTOR3 vecCoord;
     DWORD dwColor;
     FLOAT fScale;

     PRTICLE()
     {
            ZeroMemory(this,sizeof(PRTICLE));
            dwColor=0xffffffff;
            fScale=1.0f;
     }
};
```

座標、色、サイズの情報を持つ単純な型ですが、パーティクルとして初期化する際にコードが煩雑にならないように、コンストラクタを実装し、そこである程度の初期化を自分自身でするようにしました。

その型のインスタンスをMAX_PARTICLES個だけ生成します。MAX_PARTICLESは適当に300個としました。

```
PRTICLE Particle[MAX_PARTICLES+1];
```

初期位置を適当に散らせるということを行っています。難しい動きではなく、シンプルなものにしました。あくまでも、ポイントスプライトの1つの運用例です。

```
VOID InitParticles(PRTICLE* pParticles)
{
    for(DWORD i=0;i<MAX_PARTICLES;i++)
    {
          pParticles[i].vecCoord.x=( (FLOAT)rand() /(FLOAT)RAND_MAX * 4 )-
2;// -2 ～ 2
          pParticles[i].vecCoord.y=( (FLOAT)rand() /(FLOAT)RAND_MAX * 4 )-
2;// -2 ～ 2
          pParticles[i].vecCoord.z=( (FLOAT)rand() /(FLOAT)RAND_MAX * 40
)+30;// 30 ～ 70

          pParticles[i].fScale= (FLOAT)rand() /(FLOAT)RAND_MAX +0.1;// 0.1
～ 1.1
    }
}
```

毎フレーム、座標を更新する関数も必要です。次のような簡単なものとしました。

単純にZ値を減少させて（つまり、こちらに向かってくる）、Z値はゼロになったら、Zを70として、また奥に押しやるというものです。

```
VOID MoveParticle(PRTICLE* pParticle)
{
    pParticle->vecCoord.z-=0.3;
    if(pParticle->vecCoord.z<=0)
    {
          pParticle->vecCoord.z=50;
    }
    pParticle->vecCoord.x+=(FLOAT)rand() / (FLOAT)RAND_MAX /(FLOAT)10 -
0.05;//増分は-0.05 ～  0.05
    pParticle->vecCoord.y+=(FLOAT)rand() / (FLOAT)RAND_MAX/(FLOAT)10 -
0.05;//増分は-0.05 ～  0.05
}
```

ポイントスプライトのレンダリングは、次のレンダリングステートで実現します。

```
pDevice->SetRenderState( D3DRS_POINTSPRITEENABLE ,TRUE );
pDevice->SetRenderState( D3DRS_POINTSCALEENABLE,  TRUE );
```

頂点が300個あるので、あまり大きくレンダリングすると重たくなる恐れがあるので、念のため、最大サイズを32ピクセルに制限しておきます。

```
pDevice->SetRenderState( D3DRS_POINTSIZE_MAX,  GET_ADDRESS(32) );
```

ライトが有効だとレンダリングが見えないのでオフにします。

```
pDevice->SetRenderState(D3DRS_LIGHTING,FALSE);
```

深度バッファが有効だと、綺麗にアルファブレンドされないのでオフにします。

```
pDevice->SetRenderState( D3DRS_ZWRITEENABLE, FALSE );
```

あとは、レンダリングお決まりのコードです、

```
//ポイントスプライト用のテクスチャをセット
pDevice->SetTexture(0,pTexture);
//頂点の型をセット
pDevice->SetFVF(D3DFVF_XYZ |  D3DFVF_DIFFUSE | D3DFVF_PSIZE);
//レンダリング　今回は2つのポイントスプライトのみ
pDevice->DrawPrimitiveUP(D3DPT_POINTLIST, MAX_PARTICLES, Particle,
sizeof(PRTICLE));
```

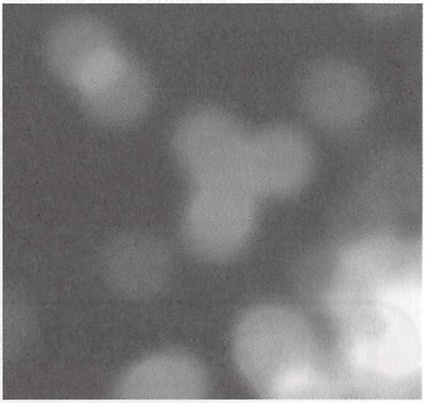
なお、このサンプルでは見栄えを良くするために、各パーティクルを透明にし、重なった部分は加算合成となるようにしました。

次のようにステートを設定します。

```
pDevice->SetRenderState(D3DRS_ALPHABLENDENABLE,TRUE);
pDevice->SetRenderState(D3DRS_SRCBLEND,D3DBLEND_ONE);
pDevice->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_ONE);
```

この効果は、右の図のようになります。

▼実行結果



One Point

パーティクルのサイズ（ピクセルサイズ）は変化しないので、サイズをも変化させたい場合は、ポイントスプライトではなく、ビルボード（ビルボーディングしたテクスチャードクアッド）を使用するようにしたほうがいいかもしれません。

シェーディングモデル

122 メッシュの頂点のみレンダリングする

●フィルモード（塗りつぶしモード）を“ポイント”に変更する　初級

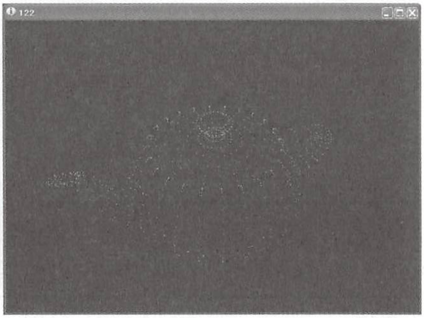
サンプルプロジェクト　122 ポイントモード

厳密に言えば、これはシェーディングモデルはありませんが、便宜上この節に含めておきます。

メッシュの頂点のみレンダリングする場合は、「フィルモード」を変更します。フィルとは（ポリゴンを）どのように塗りつぶすかという意味です。

フィルモードをポイントモードに変更することはいたってシンプルであり、次の1行で実現できます。

▼実行結果



```
pDevice->SetRenderState(D3DRS_FILLMODE,D3DFILL_POINT);
```

これだけです。

One Point

ポイントの場合、結果が見え辛いので、マテリアルにより自己発光を最大、色を白、背景を黒にして、頂点をくっきり浮き立たせるようにしました。

123 メッシュをワイヤーフレームでレンダリングする

●フィルモード（塗りつぶしモード）を“ワイヤーフレーム”に変更する　初級

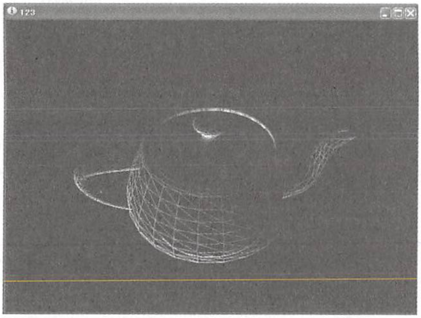
サンプルプロジェクト　123　ワイヤーフレームモード

厳密に言えば、これはシェーディングモデルはありませんが、便宜上この節に含めておきます。

メッシュの線のみレンダリングする場合は、「フィルモード」を変更します。フィルとは（ポリゴンを）どのように塗りつぶすかという意味です。

フィルモードをワイヤーフレームモードに変更することはいたってシンプルであり、次の1行で実現できます。

▼実行結果



```
pDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
```

One Point

ポリゴンの中身を塗りつぶす場合は、D3DFILL_SOLID（ソリッドモード）とします。デフォルトのフィルモードはD3DFILL_SOLIDです。

124 フラットシェーディングする

●シェーディングモードを“フラット”に変更する

初級

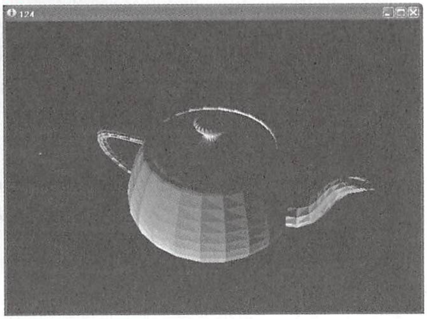
サンプルプロジェクト　124 フラットシェーディング

フラットシェーディングはそもそもポリゴン表面の陰影を全く計算せずに、単色で単純に塗りつぶすシェーディングモデルです。ただ、Direct3Dの場合、ライトを設置していると、フラットシェーディングでもライティング計算を行うので、不自然ではありますが、陰影が現われます。

このサンプルではライトを1つ設置しているので、図のようにライトの方向が分かるような陰影が出ています。

シェーディングモードをフラットシェーディングに変更することはいたってシンプルであり、次の1行で実現できます。

▼実行結果



```
pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_FLAT);
```

One Point

フラットシェーディングは見栄えが悪い代わりに、シェーディングモデルの中で最も高速なモードです。

125 グーローシェーディングする

シェーディングモードを“グーロー”に変更する

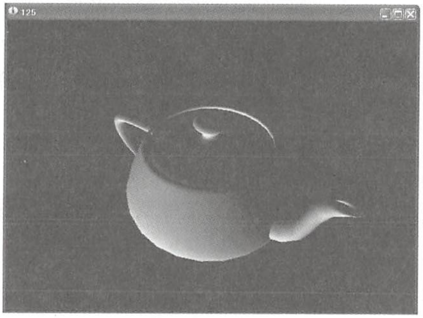
初級

サンプルプロジェクト　125　グーローシェーディング

グーローシェーディングは各頂点の頂点法線とライトの情報を基に、どの程度シェーディング（陰影付け）を行うかを計算します。そのため、全体として滑らかな陰影が付くのでフラットシェーディングと比べて、見栄えが良くなります。

シェーディングモードをグーローシェーディングに変更することはいたってシンプルであり、次の1行で実現できます。

▼実行結果



```
pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_GOURAUD);
```

One Point

D3DSHADE_GOURAUDはデフォルトなので、指定しなくても通常はグーローモードでシェーディングされます。

126 フォンシェーディングする

●レンダリングステートでスペキュラを有効にする

初級

サンプルプロジェクト　126 フォンシェーディング

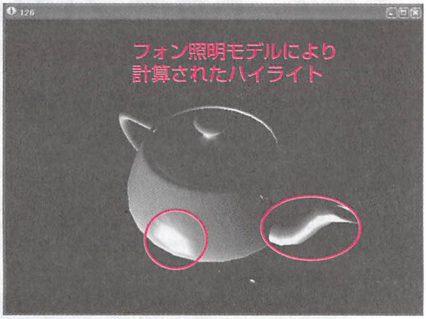
Direc3Dにおいてフォンシェーディングはサポートされていません。

しかし、ライトの計算には簡易版のフォン照明モデルを採用していますので、結果として出力されるものはフォンシェーディングと似ています。

ただし、ライトを最低一個設置し、ライトのプロパティとメッシュ側のマテリアルにおいてスペキュラ成分に適当な値を入れ、かつ、レンダリングステートをスペキュラ有効にしなくてはなりません。

なお、シェーディングモードはグーローにします。グーローは基本的に頂点単位のシェーディングなので、Tips131などで行うピクセル単位のフォンシェーディングほど綺麗に光沢は出ません。特に、ポリゴン数が少ない場合はその弱点が顕著に出てしまいます。

▼フォン照明モデルにより計算されたハイライト



●レンダリングステート設定

```
pDevice->SetRenderState(D3DRS_SPECULARENABLE,TRUE);
pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_GOURAUD);
```

●ライトの設置とライトのプロパティ設定

```
// ライトをあてる
 D3DXVECTOR3 vecDirection(1,1,1);
D3DLIGHT9 light;
ZeroMemory( &light, sizeof(D3DLIGHT9) );
light.Type        = D3DLIGHT_DIRECTIONAL;
light.Diffuse.r   = 1.0f;
light.Diffuse.g   = 1.0f;
light.Diffuse.b   = 1.0f;
light.Specular.r=1.0f;
light.Specular.g=1.0f;
light.Specular.b=1.0f;
```

```
D3DXVec3Normalize( (D3DXVECTOR3 *)&light.Direction, &vecDirection );
light.Range          = 200.0f;
pDevice->SetLight( 0, &light );
pDevice->LightEnable( 0, TRUE );
```

One Point

完全なフォンシェーディングはプログラマブルシェーダーにより実現します。それについてはTips131で解説します。

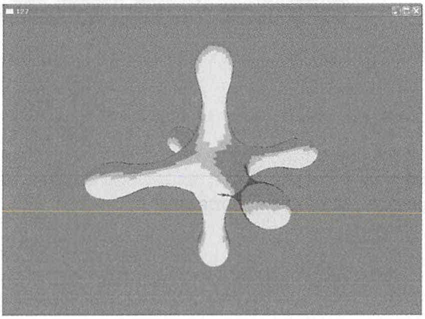
127 トゥーン（セル画風）シェーディングする

●プログラマブルシェーダーでトゥーンレンダラを作成する

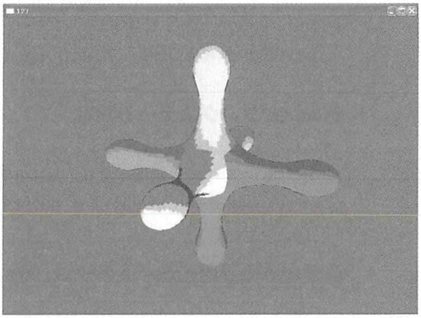
上級

サンプルプロジェクト　127　トゥーンシェーディング
応用サンプルプロジェクト　127-2　トゥーンシェーディング（テクスチャ付き）

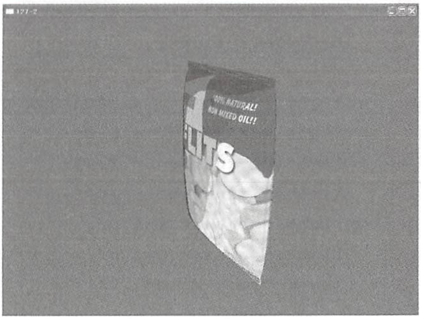
▼実行結果



▼実行結果



▼実行結果



●テクスチャの色成分は計算しないバージョン

トゥーンシェーディングは、固定機能のシェーダーには実装されていないシェーディングモデルなので、プログラマブルシェーダーを自分でプログラミングして実現するしかありません。

原理は非常に上手くできています。にもかかわらずシンプルなので理解は容易です。たった4ピクセルのテクスチャ（というよりも

超コンパクトな“濃淡マップ”）を利用します。

まず輪郭ですが、視線ベクトルとライトベクトルの角度を判断すれば、輪郭を作ることができます。具体的には2つのベクトルを正規化し内積をとれば角度が得られます。輪郭付近のポリゴンは視線に対しより垂直なので内積の値がゼロに近くなり、物体の内側になるほど内積が1に近づくというわけです。ゼロに近いポリゴンは黒に、それ以外は白（あるいはグレー）にすることにより輪郭となります。

そして内側のシェーディングですが、これも同様に視線ベクトルとライトベクトルの内積を計算します。輪郭の場合と異なるのは、テクスチャです。輪郭の場合は、はっきりと“黒か白”のテクスチャですが、シェーディングの場合は、“黒から白への数段階のグラデーション”になります。まぁグラデーションと言っても、4ピクセルしか無いのですから、最高でも濃淡の変化は4段階にすることしかできず、滑らかなグラデーションではありません。逆に言うと、滑らかになってしまっては困ります。濃淡の段階を多くして滑らかにしてしうのなら、なにも苦労してプログラマブルシェーダーを使う意味がなくなります。ある意味、濃淡の段階がすくないほどセル画に近くなり、それがピクセル数を4ピクセルと極端に少なくした理由です。したがって、ピクセル数をたとえば8とか16にしたのなら、より滑らかなシェーデングになりますが、あまり滑らかにしないようにしましょう。

本サンプルでは、

内積が0～0.25なら、濃いグレー
内積が0.25～0.5なら、グレー
内積が0.5～0.75なら、淡いグレー
内積が0.75～1なら、白
となります。

ポリゴンのマテリアル色（正確にはディフューズ値）に、それらのグレーを掛け合わせて最終色とすれば、濃淡の完成です。

●シェーダーファイル（ToonShader.fx）

```
//////////////////////////////
//グローバル
//////////////////////////////
float4x4 matProj;
float4x4 matView;
float4x4 matWorld;
float3 LightPos;
float3 EyePos;
float4 Diffuse;
texture ShadeTexture;
texture LineTexture;
//////////////////////////////
//定義
//////////////////////////////
sampler ShadeSampler = sampler_state
{
    Texture = (ShadeTexture);
};

sampler LineSampler = sampler_state
```

```
{
    Texture = (LineTexture);
};

struct VS_OUT_PS_IN
{
    float4 Position : POSITION;
    float4 Color : COLOR;
    float Line : TEXCOORD;
    float Shade : TEXCOORD1;
};
//////////////////////////////
//バーテックス・シェーダー
//////////////////////////////
VS_OUT_PS_IN VS(float3 Position : POSITION, float3 Normal : NORMAL, float4
Color : COLOR)
{
    VS_OUT_PS_IN Out = (VS_OUT_PS_IN)0;

    float3 lightVec = normalize(LightPos - Out.Position);
    float3 eyeVec = normalize(EyePos - Out.Position);
    Normal = mul(Normal, (float3x3)matWorld);
    Out.Position = mul(float4(Position, 1), matWorld);
    Out.Line = max(dot(Normal, eyeVec), 0);
    Out.Shade = max(dot(Normal, lightVec), 0);
    Out.Position = mul(Out.Position, mul(matView, matProj));
    Out.Color = Color;

    return Out;
}
//////////////////////////////
//ピクセル・シェーダー
//////////////////////////////
float4 PS( VS_OUT_PS_IN In) : COLOR
{
    float4 Color;

    Color =  Diffuse * tex1D(LineSampler, In.Line) * tex1D(ShadeSampler,
In.Shade);

    return Color;
}
//////////////////////////////
// テクニック
//////////////////////////////
```

```
technique ToonShading
{
    pass Toon
    {
            Sampler[0] = (LineSampler);
            Sampler[1] = (ShadeSampler);

            VertexShader = compile vs_2_0 VS();
            PixelShader = compile ps_2_0 PS();
    }
}
```

●実装ファイル(Entry.cpp)

濃淡マップ（シェーディング用、陰影用）の画像ファイルからテクスチャを作成します。

```
if( FAILED( D3DXCreateTextureFromFile( pDevice, "Shade.bmp",
                                        &pTextureShade ) ) )
{
    MessageBox(NULL, "テクスチャの読み込みに失敗しました", NULL, MB_OK);
}
```

濃淡マップ（輪郭用）の画像ファイルからテクスチャを作成します。

```
if( FAILED( D3DXCreateTextureFromFile( pDevice, "Outline.bmp",
                                        &pTextureLine ) ) )
{
    MessageBox(NULL, "テクスチャの読み込みに失敗しました", NULL, MB_OK);
}
```

シェーダーファイルを読み込みます。

```
//シェーダーを読み込む
if( FAILED(  D3DXCreateEffectFromFile(pDevice, "ToonShader.fx", NULL, NULL,
    0, NULL, &g_pEffect, NULL)))
{
    MessageBox( NULL, "シェーダーファイル読み込み失敗", "", MB_OK);
    return E_FAIL;
}
```

Zバッファーを有効にします。

```
// zバッファー処理を有効にする
    pDevice->SetRenderState( D3DRS_ZENABLE, TRUE );
```

●レンダリング部分

```
//レンダリング
    if( SUCCEEDED( pDevice->BeginScene() ) )
    {
        g_pEffect->SetTechnique( "ToonShading" );
        D3DXMATRIX matAll = matWorld ＊ matView ＊ matProj;
        g_pEffect->SetMatrix( "matProj", &matProj );
        g_pEffect->SetMatrix( "matView", &matView );
        g_pEffect->SetMatrix( "matWorld", &matWorld );
        g_pEffect->SetTexture("ShadeTexture",pTextureShade);
        g_pEffect->SetTexture("LineTexture",pTextureLine);

        g_pEffect->SetVector("LightPos",&D3DXVECTOR4(0,0,-10,1));
        g_pEffect->SetVector("EyePos",(D3DXVECTOR4＊)&vecEyePt);

        g_pEffect->Begin( NULL, 0 );

        for( DWORD i=0; i<dwNumMaterials; i++ )
        {
            g_pEffect->BeginPass(0);
            g_pEffect->SetFloatArray("Diffuse",(FLOAT
＊)&pMeshMaterials[i].Diffuse,4);
            pMesh->DrawSubset( i );
            g_pEffect->EndPass();
        }

        g_pEffect->End();
        pDevice->EndScene();
    }
    pDevice->Present( NULL, NULL, NULL, NULL );
```

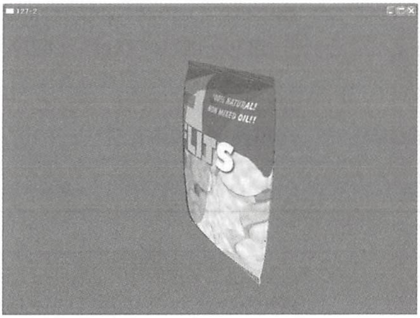
●テクスチャを考慮したバージョン

直前のコードは、テクスチャを考慮せず、メッシュのマテリアルのみですが、メッシュにテクスチャが貼られている場合は、多少手を加える必要があります。

ただ、基本原理は全く同じで、しかも、手を加える部分はほんのわずかです。具体的には、単に、最終色にテクスチャの色成分を掛け合わせるだけです(ほんとうにわずかですね)。

テクスチャなしのコードと異なるキーポイントだけコードを載せます（便宜上、シェーダーファイルは全てのコードを載せます)。

▼実行結果



●シェーダーファイル（ToonShader.fx） テクスチャも考慮したバージョン

```
//////////////////////////////
//グローバル
//////////////////////////////
float4x4 matProj;
float4x4 matView;
float4x4 matWorld;
float3 LightPos;
float3 EyePos;
float4 Diffuse;
texture DecalTexture;
texture ShadeTexture;
texture LineTexture;
//////////////////////////////
//定義
//////////////////////////////
sampler DecalSampler = sampler_state
{
    Texture = (DecalTexture);
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = NONE;
    AddressU = Clamp;
    AddressV = Clamp;
};

sampler ShadeSampler = sampler_state
{
    Texture = (ShadeTexture);
};

sampler LineSampler = sampler_state
{
    Texture = (LineTexture);
};

struct VS_OUT_PS_IN
{
    float4 Position : POSITION;
    float4 Color : COLOR;
    float2 Decal : TEXCOORD;
    float Line : TEXCOORD1;
    float Shade : TEXCOORD2;
```

```
};
//////////////////////////////
//バーテックス・シェーダー
//////////////////////////////
VS_OUT_PS_IN VS(float3 Position : POSITION, float3 Normal : NORMAL, float4
Color : COLOR,float2 TexCoord :TEXCOORD)
{
    VS_OUT_PS_IN Out = (VS_OUT_PS_IN)0;

    float3 lightVec = normalize(LightPos - Out.Position);
    float3 eyeVec = normalize(EyePos - Out.Position);
    Normal = mul(Normal, (float3x3)matWorld);
    Out.Position = mul(float4(Position, 1), matWorld);
    Out.Line = max(dot(Normal, eyeVec), 0);
    Out.Shade = max(dot(Normal, lightVec), 0);
    Out.Position = mul(Out.Position, mul(matView, matProj));
    Out.Color = Color;
    Out.Decal = TexCoord;

    return Out;
}
//////////////////////////////
//ピクセル・シェーダー
//////////////////////////////
float4 PS( VS_OUT_PS_IN In) : COLOR
{
    float4 Color;

    Color = tex2D(DecalSampler , In.Decal) * Diffuse * tex1D(LineSampler,
In.Line) * tex1D(ShadeSampler, In.Shade);

    return Color;
}
//////////////////////////////
// テクニック
//////////////////////////////
technique ToonShading
{
    pass Toon
    {
        Sampler[0] = (LineSampler);
        Sampler[1] = (ShadeSampler);

        VertexShader = compile vs_2_0 VS();
```

```
            PixelShader = compile ps_2_0 PS();
        }
    }
```

●実装ファイル(Entry.cpp)　デカールとなるテクスチャをセットする

```
 for( DWORD i=0; i<dwNumMaterials; i++ )
 {
     g_pEffect->BeginPass(0);
      g_pEffect->SetFloatArray("Diffuse",(FLOAT
*)&pMeshMaterials[i].Diffuse,4);
      g_pEffect->SetTexture("DecalTexture",pMeshTextures[i]);
     pMesh->DrawSubset( i );
     g_pEffect->EndPass();
 }
```

One Point

ここでは、濃淡マップを画像ファイルから得ていますが、たかだか数ピクセルであれば、なにもファイルから取得することはなく、コード内に直接値を書いたほうが、スッキリするでしょう。ただ、コードが多少なりとも煩雑になるので、外部ファイル（ビットマップファイル）を濃淡マップに利用しました。

陰影と光沢

128 固定機能シェーダーで陰影を付ける

●ライトを1個以上設置して陰影を出す

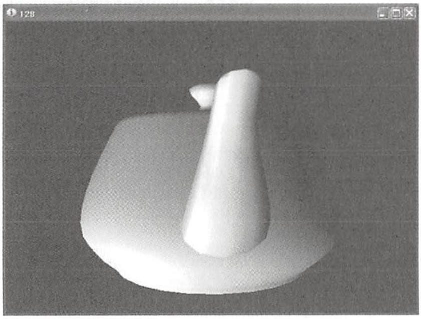
初級

サンプルプロジェクト　128　固定機能で陰影を出す

今までのメッシュをレンダリングするサンプルのほとんどは、ライトを設置していたので、陰影は出ています。固定機能における陰影は、特別なことではなく、ある意味当たり前のことかもしれません。

Direct3Dに限らず陰影を出すには（光沢を出す際にも）、ライティング計算が必要です。Direct3Dでは、次のコードによりパイプラインにライティング計算をさせなければ陰影は出ません。

▼実行結果



```
pDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
```

さらに、ライトを一個以上作成しなければなりません。“ライト”とはコード的には、D3DLIGHT9のインスタンスです。

実際のコードは次のようになります。

```
D3DLIGHT9 light;
ZeroMemory( &light, sizeof(D3DLIGHT9) );
light.Type          = D3DLIGHT_DIRECTIONAL;
light.Diffuse.r    = 1.0f;
light.Diffuse.g    = 1.0f;
light.Diffuse.b    = 1.0f;
light.Direction=D3DXVECTOR3(1,1,1);
light.Range          = 200.0f;
pDevice->SetLight( 0, &light );
pDevice->LightEnable( 0, TRUE );
```

（※各メンバに入れる値は自由です）

One Point

陰影がなく、ベタっとしたレンダリングになってしまっている場合は、ライトを設定しているか確認しましょう。

129 プログラマブルシェーダーで陰影を付ける

ランバート照明モデルを自分で実装する

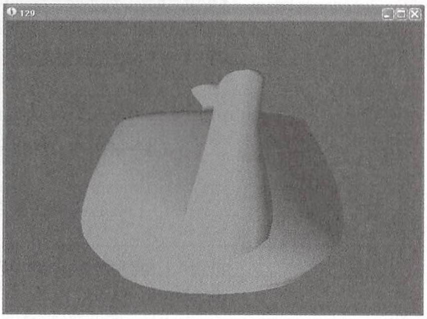
上級

サンプルプロジェクト　129　プログラマブルシェーダーで陰影を出す

ランバート（ランベルトとも言います）の余弦則（Lambert Law）に則った照明モデルでランバート照明モデル（Lambert Illumination Model）というものがあります。

リアルタイム3DCGの陰影はほとんどこのランバートモデルにより実装されます。ランバートモデルは非常にシンプルです。そもそも陰影の原理及び実装はごく単純なので、1つの照明モデルで間に合っている感があります。

▼実行結果



●Main.cpp側

```
LPD3DXEFFECT pEffect=NULL;
```

●Main.cpp側

```
//シェーダーを読み込む
if( FAILED(  D3DXCreateEffectFromFile(pDevice, "LambertShader.fx", NULL,
NULL,
    0, NULL, &pEffect, NULL)))
{
    MessageBox( NULL, "シェーダーファイル読み込み失敗", "", MB_OK);
    return E_FAIL;
}
```

●Main.cpp側

```
//ティーポットのレンダリング
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //エフェクト（シェーダー）のグローバル変数に値を渡す
    {
            //ワールド・ビュー・プロジェクション行列を渡す
```

```
        pEffect->SetMatrix("WVP",(D3DXMATRIX＊)&(matWorld＊matView＊
matProj));
        //ワールド行列の逆行列の転置行列を渡す
        D3DXMATRIX matWIT;        pEffect->SetMatrix("WIT",
D3DXMatrixTranspose(&matWIT,D3DXMatrixInverse(&matWIT,NULL,&matWorld))
);
        //ライトの方向ベクトルを渡す
        pEffect->SetVector("LightDir",&D3DXVECTOR4(-1,-1,-1,0));
        //入射光（ライト）の強度を渡す　目一杯明るい白色光にしてみる
        pEffect->SetVector("LightIntensity",&D3DXVECTOR4(1,1,1,1));
        //拡散反射率を渡す 赤にしてみた
        pEffect->SetVector("Diffuse",&D3DXVECTOR4(1,0,0,1));
    }
        pEffect->Begin( NULL, 0 );
            pEffect->BeginPass(0);

                pMesh->DrawSubset(0);

            pEffect->EndPass();
        pEffect->End();

    pDevice->EndScene();
}
```

●Lambert.fx（シェーダーファイル）

```
///////////////////////////////
//型定義
///////////////////////////////

struct VS_OUT
{
    float4 Position     : POSITION;
    float4 Color        : COLOR0;
};

///////////////////////////////
//グローバル
///////////////////////////////
float4x4 WVP; //ワールド行列
float4x4 WIT;  //ワールド行列の逆行列の転置行列
float4 LightDir;   //r ライトの方向ベクトル
float4 LightIntensity; //入射光の強度
float4 Diffuse; //拡散反射率
```

```
////////////////////////////
//バーテックス・シェーダー
////////////////////////////
VS_OUT VS(float4 Position : POSITION,float3 Normal : NORMAL)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Position=mul(Position,WVP);

    float3 WNormal=normalize(mul(Normal,(float3x3)WIT));

    Out.Color=LightIntensity * Diffuse * dot(WNormal,LightDir);//この式はランバートの余弦則だけですが、通常、ランバート照明モデルと言えば環境光の項を含めることが多いようです

    return Out;
}
///////
//ピクセル・シェーダー
///////
float4 PS(VS_OUT In) : COLOR
{
    return In.Color;
}

////////////////////////////
// テクニック
////////////////////////////
technique tecLambert
{
    pass Lambert
    {
        VertexShader = compile vs_2_0 VS();
        PixelShader  = compile ps_2_0 PS();
    }
}
```

なお、本サンプルのシェーダーは、照明モデルとしては不完全です。不完全と言っても、環境光（アンビエントライト）の項を省いただけです。

環境光を含めたい場合は、シェーダーファイル側にfloat4 Ambient;などとグローバル変数を用意し、それにアプリケーション側から適当な値を入れ、シェーダー側の
Out.Color=LightIntensity ＊ Diffuse ＊ dot(WNormal,LightDir);
の部分を
Out.Color=Ambient ＊ LightIntensity ＊ Diffuse ＊ dot(WNormal,LightDir);
などとすればいいだけのことです。

One Point

シェーダーファイルではライトやマテリアルの計算をするので、固定機能のようにライトの作成・レンダリングステートでのライト設定・物体側のマテリアル設定等が不要になり、考えようによってはコードがスッキリします。

130 固定機能シェーダーで光沢を付ける

●ライトを一個以上作成し、ライトとメッシュ両方のスペキュラ値を設定する　中級

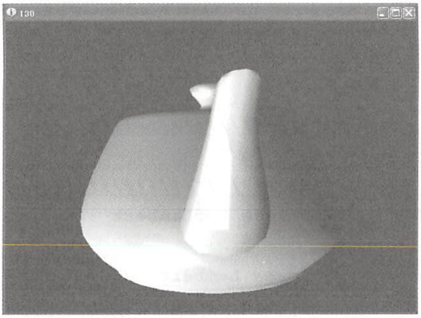
サンプルプロジェクト　130　固定機能で光沢を付ける

ここでは固定機能シェーダーを使用します。したがって本質的にはグーローシェーディング（頂点単位）なので、光沢の付き方はポリゴンの細かさに依存します。光沢が、Tips131の光沢とは異なることが分かるでしょうか？

固定機能の光沢のほうが、角ばっています。これはグーローシェーディングが頂点ベースで光沢を計算しているからです。固定機能シェーダーであってもポリゴンがより細かければ、結果は異なります。見方によっては、このほうがジオメトリの形状に忠実なテカリになるということもできるでしょう。

さて、光沢をつけるには、まず何と言ってもライトの設置と設定が最低条件です。陰影を出すにもライトが必須でしたが、光沢の場合は、それにもう一手間かけます。具体的にはライトプロパティとメッシュのマテリアル両方のスペキュラ成分を設定することです。

▼実行結果



●物体側（メッシュ）のマテリアル

```
//ティーポットメッシュのマテリアル
    pMeshMaterials = new D3DMATERIAL9;
    pMeshMaterials->Diffuse.r=1.0f;
    pMeshMaterials->Diffuse.g=1.0f;
    pMeshMaterials->Diffuse.b=1.0f;
    pMeshMaterials->Diffuse.a=1.0f;
    pMeshMaterials->Ambient=pMeshMaterials->Diffuse;
    pMeshMaterials->Specular.r=1.0f;
    pMeshMaterials->Specular.g=1.0f;
    pMeshMaterials->Specular.b=1.0f;
    pMeshMaterials->Emissive.r=0.1f;
    pMeshMaterials->Emissive.g=0.1f;
    pMeshMaterials->Emissive.b=0.1f;
    pMeshMaterials->Power=120.0f;
```

●ライトの作成とプロパティ設定

```
// ライトを設置
     D3DLIGHT9 light;
     ZeroMemory( &light, sizeof(D3DLIGHT9) );
     light.Type        = D3DLIGHT_DIRECTIONAL;
     light.Diffuse.r   = 1.0f;
     light.Diffuse.g   = 1.0f;
     light.Diffuse.b   = 1.0f;
     light.Specular.r=1.0f;
     light.Specular.g=1.0f;
     light.Specular.b=1.0f;
     light.Direction=D3DXVECTOR3(1,1,1);
     light.Range         = 200.0f;
     pDevice->SetLight( 0, &light );
     pDevice->LightEnable( 0, TRUE );
```

●レンダリング

```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //ティーポットのレンダリング

    // ライトをオンにする
    pDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
    // スペキュラ（鏡面反射）を有効にする
    pDevice->SetRenderState(D3DRS_SPECULARENABLE,TRUE);

    pDevice->SetMaterial(pMeshMaterials);
    pMesh->DrawSubset(0);

    pDevice->EndScene();
}
```

One Point

固定機能の場合、ライトと物体という概念が明確に分かれています。また、それぞれのスペキュラを設定せずに片方だけを設定しても効果が出ません。

131 プログラマブルシェーダーで光沢を付ける

●フォン照明モデルを自分で実装する

上級

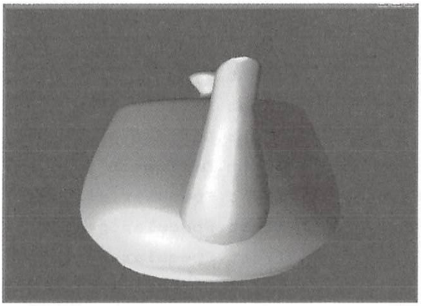
サンプルプロジェクト 131 プログラマブルシェーダーで光沢を付ける

フォンシェーディングは、頂点間を補間するシェーディングモデルで、ピクセル単位にフォン照明モデルを適用した状態に近くなります。

ところが残念なことにDirect3Dの固定機能シェーダーはフォンシェーディングができません。おそらくパフォーマンス対効果を考えてのことでしょう。

さて、本サンプルはプログラマブルシェーダーを使用しています。ライティング計算はピクセル単位なので、メッシュの頂点数に依存しない滑らかな光沢を作ることができます。

▼実行結果



●Main.cpp側

```
//シェーダーを読み込む
if( FAILED(  D3DXCreateEffectFromFile(pDevice, "PhongShader.fx", NULL, NULL,
    0, NULL, &pEffect, NULL)))
{
    MessageBox( NULL, "シェーダーファイル読み込み失敗", "", MB_OK);
    return E_FAIL;
}
```

●Main.cpp側

```
//ティーポットのレンダリング
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //エフェクト（シェーダー）のグローバル変数に値を渡す
    {
          //ワールド・ビュー・プロジェクション行列を渡す
          pEffect->SetMatrix("matWVP",(D3DXMATRIX*)&(matWorld*matView*
matProj));
          //ワールド行列を渡す
          pEffect->SetMatrix("matWorld",&matWorld);
          //ライトの方向ベクトルを渡す
          pEffect->SetVector("vecLightDir",&D3DXVECTOR4(-1,-1,-1,0));
          //視点位置（カメラ位置）を渡す
          pEffect->SetVector("vecEye",(D3DXVECTOR4＊)&vecEyePt);
          //拡散反射率を渡す　水色にしてみた
          pEffect->SetVector("Diffuse",&D3DXVECTOR4(0,1,1,1));
    }
}
```

●PhongShading.fx側

```
/////////////////////////////
//型定義
/////////////////////////////
struct VS_OUT
{
    float4 Position : POSITION;
    float3 Light  : TEXCOORD0;
    float3 Normal  : TEXCOORD1;
    float3 View  : TEXCOORD2;
};

/////////////////////////////
//グローバル
/////////////////////////////
float4x4 matWVP;
float4x4 matWorld;
float4 vecLightDir;
float4 vecEye;
float4 Diffuse={0,0,0,0};

/////////////////////////////
//バーテックス・シェーダー
/////////////////////////////
VS_OUT VS(float4 Pos : POSITION, float3 Normal : NORMAL)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Position = mul(Pos, matWVP);
    Out.Light = vecLightDir;
    float3 PosWorld = normalize(mul(Pos, matWorld));
    Out.View = vecEye - PosWorld;
    Out.Normal = mul(Normal, matWorld);

    return Out;
}
///////////////////////////////
//ピクセル・シェーダー
////////////////////////////////
float4 PS(VS_OUT In) : COLOR
{
    float3 Normal = normalize(In.Normal);
    float3 LightDir = normalize(In.Light);
    float3 ViewDir = normalize(In.View);
    float4 NL = saturate(dot(Normal, LightDir));

    float3 Reflect = normalize(2 * NL * Normal - LightDir); //反射ベクトルの公式通り
    float4 specular = pow(saturate(dot(Reflect, ViewDir)), 4);

    return Diffuse * NL + specular; //通常、照明モデルと言えば環境光の項を含めますが、簡単のため除きました。
}

///////////////////////////////
// テクニック
///////////////////////////////
technique tecPhong
{
    pass Phong
    {
        VertexShader = compile vs_2_0 VS();
        PixelShader  = compile ps_2_0 PS();
    }
}
```

132 アイススティックの表面のような霜(しも)を付ける

視線ベクトルと頂点法線ベクトルの内積で作る

上級

サンプルプロジェクト　132　表面に霜を付ける

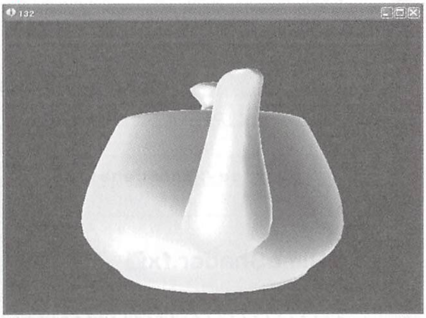
　このシェーダーはメッシュの表面にうっすら霜がかぶさっているようなレンダリングをします。アイスキャンディを想像してもらえれば、それに似ています（紙面では分かりま

せんが、ソーダアイスです）。

原理は、陰影をつける場合における視線と法線の内積の補数を利用することにより実現しています。

このシェーダーを公開するのは本書が世界最初です。ちなみに筆者は、ノースブレイン（筆者の個人屋号）のロゴレンダリングにそっくりなこのシェーダーをノースブレイン・シェーダーと読んでいます。一般的にはライム（霜）シェーダーと呼んでもらっても構いません。

▼実行結果



●Main.cpp側

```
//シェーダーを読み込む
if( FAILED(  D3DXCreateEffectFromFile(pDevice, "RimeShader.fx", NULL, NULL,
    0, NULL, &pEffect, NULL)))
{
    MessageBox( NULL, "シェーダーファイル読み込み失敗", "", MB_OK);
    return E_FAIL;
}
```

●Main.cpp側

```
//ティーポットのレンダリング
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //エフェクト（シェーダー）のグローバル変数に値を渡す
    {
            //ワールド・ビュー・プロジェクション行列を渡す
            pEffect->SetMatrix("matWVP",(D3DXMATRIX*)&(matWorld*matView*
matProj));
            //ワールド行列を渡す
            pEffect->SetMatrix("matWorld",&matWorld);
            //ライトの方向ベクトルを渡す
            pEffect->SetVector("vecLightDir",&D3DXVECTOR4(-1,-1,-1,0));
            //視点位置（カメラ位置）を渡す
            pEffect->SetVector("vecEye",(D3DXVECTOR4*)&vecEyePt);
            //拡散反射率を渡す　水色にしてみた
            pEffect->SetVector("Diffuse",&D3DXVECTOR4(0,1,1,1));
    }
            pEffect->Begin( NULL, 0 );
                    pEffect->BeginPass(0);
```

```

                      pMesh->DrawSubset(0);

                pEffect->EndPass();
          pEffect->End();

    pDevice->EndScene();
}
```

●RimeShader.fx側

```
//////////////////////////////
//型定義
//////////////////////////////
struct VS_OUT
{
    float4 Position : POSITION;
    float3 Light : TEXCOORD0;
    float3 Normal : TEXCOORD1;
    float3 View : TEXCOORD2;
};

//////////////////////////////
//グローバル
//////////////////////////////
float4x4 matWVP;
float4x4 matWorld;
float4 vecLightDir;
float4 vecEye;
float4 Diffuse=float4(0,0,0,0);

//////////////////////////////
//バーテックス・シェーダー
//////////////////////////////
VS_OUT VS(float4 Pos : POSITION, float3 Normal : NORMAL)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Position = mul(Pos, matWVP);
    Out.Light = vecLightDir;
    float3 PosWorld = normalize(mul(Pos, matWorld));
    Out.View = vecEye - PosWorld;
    Out.Normal = mul(Normal, matWorld);

    return Out;
}
//////////////////////////////
//ピクセル・シェーダー
//////////////////////////////
float4 PS(VS_OUT In) : COLOR
{
    float3 Normal = normalize(In.Normal);
    float3 LightDir = normalize(In.Light);
    float3 ViewDir = normalize(In.View);
    float4 NL = saturate(dot(Normal, LightDir));

    float3 Reflect = normalize(2 * NL * Normal - LightDir); //反射ベクトルの公式通り
    float4 specular = pow(saturate(dot(Reflect, ViewDir)), 2);

    float4 VN=dot(ViewDir,Normal);
    float4 Rime=pow(float4(1,1,1,1)*(1-VN),1);//Rime：表面の霜（しも）

    return Diffuse * NL + specular +Rime; //通常、照明モデルと言えば環境光の項を含めますが、簡単のため除きました。
}

//////////////////////////////
// テクニック
//////////////////////////////
technique tecRime
{
    pass p0
    {
        VertexShader = compile vs_2_0 VS();
        PixelShader  = compile ps_2_0 PS();
    }
}
```

One Point

“霜”としましたが、値を調整すればもっとテカリのある“ワックス”のような感じにすることもできるでしょう。

マテリアル（色、光沢等）

133 マテリアルとは

●ジオメトリの表面上での光の反射具合のこと

初級

現実世界において物体が目に見えるのは、光（ライト）が反射するからであり（物体に反射した光が目に入るからであり）、光がなければ何も見ることができません（自己発光するものは除く）。Direct3Dにおいて、ライティングをオフにしても（光がなくても）レンダリング結果が見えるのは、ある意味、現実に反したCG特有のものであることを認識することが大切です。CG特有のライトとは具体的には「アンビエント光」によりレンダリングです。これはDirect3Dに限らず3DCG全般に当てはまります。

さてマテリアルは、ジオメトリの表面上での光の反射具合のことです。当然、ライティング計算を行ってはじめて反映されます（効果を確認できます）。ライティング計算は、レンダリングステート（ID3Device9::SetRenderState）でD3DRS_LIGHTINGをTRUEにすることにより有効になります。また、ライティング計算には法線情報が必要であり、法線により、ディフューズ反射（色と陰影）とスペキュラー反射（光沢、ハイライト）の計算が可能になります。

したがって、ジオメトリが法線を持っていないとライティング計算が行えませんので、頂点を定義する場合は法線情報を持つように定義し、そして、SetFVFにより型を通知する時は、D3DFVF_NORMALフラグを指定してやる必要があります。

マテリアルにおけるディフューズ計算によりジオメトリ表面の“陰影及び色“が決定されることになり、また、スペキュラー計算により” 光沢（ハイライト）“言い換えれば” てかり具合“が決定されます。平たく表現すると、マテリアルは” 物質表面の材質感” のパラメーターです。

Direct3Dのライトはディフューズにおいてランバートモデル、スペキュラーにおいてフォンモデル（簡易版）により反射を計算します。

134 マテリアルの作成と設定

●マテリアルのオブジェクトを作成しレンダリングに反映させる

初級

サンプルプロジェクト　134　マテリアルの作成と設定

まず、マテリアルオブジェクトのポインターを準備します。

```
D3DMATERIAL9＊ pMaterial=NULL;
```

インスタンス化し、そのポインターを代入します。

```
pMaterial = new D3DMATERIAL9;
```

各メンバを適当に初期化します。

```
pMaterial->Ambient.r=1.0f;
pMaterial->Ambient.g=1.0f;
pMaterial->Ambient.b=1.0f;

pMaterial->Diffuse.r=1.0f;
pMaterial->Diffuse.g=1.0f;
pMaterial->Diffuse.b=1.0f;
pMaterial->Diffuse.a=1.0f;

pMaterial->Specular.r=0.0f;
pMaterial->Specular.g=0.0f;
pMaterial->Specular.b=0.0f;
pMaterial->Power=0.0f;

pMaterial->Emissive.r=0.1f;
pMaterial->Emissive.g=0.1f;
pMaterial->Emissive.b=0.1f;
```

レンダリングコードよりも前に、パイプラインにセットします。

```
pDevice->SetMaterial(pMaterial);
(レンダリング)
```

プログラムを終了する際には、マテリアルをデリートします。

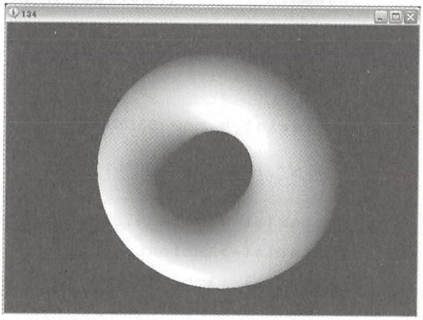
```
delete pMaterial;
```

なお、より礼儀正しいコードとしては、このようにすでにデリートしていないかどうかのチェック機能を持たせたマクロを定義して、それを使用するのがいいでしょう。

```
#define SAFE_DELETE(p) { if(p) { delete (p); (p) = NULL; } }
SAFE_DELETE( pMaterial );
```

右の図はレンダリング結果です。

▼実行結果



＊＊＊＊＊＊

Specular（スペキュラー）は“光沢”を意味するメンバです。

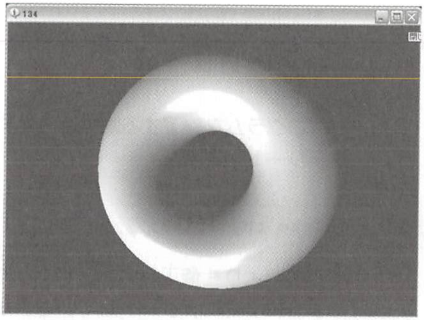
```
pMaterial->Specular.r=1.0f;
pMaterial->Specular.g=1.0f;
pMaterial->Specular.b=1.0f;
pMaterial->Power=120.0f;
```

スペキュラーの値を反映するにはパイプラインがスペキュラー計算を行うようなモードになっていなければなりません。その設定は次のコードで行います。

```
pDevice->SetRenderState(D3DRS_SPECULARENABLE,TRUE);
```

結果は右の図のようになります。

▼実行結果



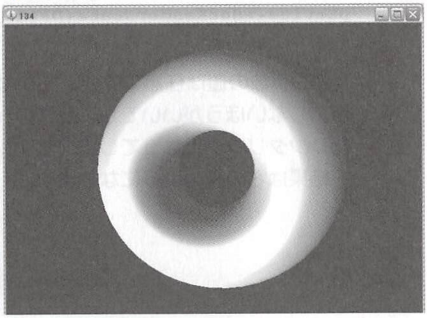
＊＊＊＊＊＊

Power（パワー）は、“光沢の度合い“を意味します。値が低ければ広範囲にテカり、逆に高ければ光沢自体の面積は少なくなります。

```
pMaterial->Power=20.0f;
```

結果は右の図のようになります。

▼実行結果



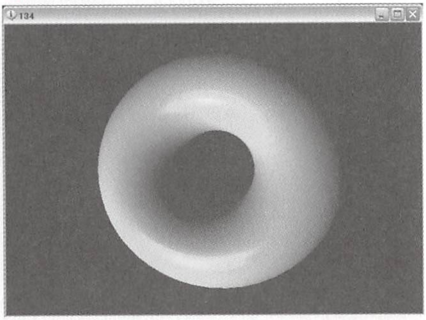
＊＊＊＊＊＊

Diffuse（ディフューズ）は“色”を意味するメンバです。

```
pMaterial->Diffuse.r=0.0f;
pMaterial->Diffuse.g=1.0f;
pMaterial->Diffuse.b=1.0f;
pMaterial->Diffuse.a=1.0f;
```

結果は右の図のようになります。

▼実行結果



＊＊＊＊＊＊

Ambient（アンビエント）は“環境光”　を意味するメンバです。
（陰の部分もお構いなしに均一に明るくなる）

```
pMaterial->Ambient.r=1.0f;
pMaterial->Ambient.g=10.0f;
pMaterial->Ambient.b=1.0f;
```

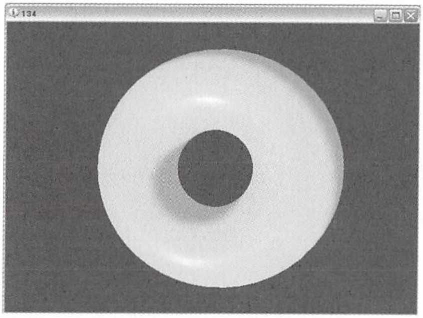
アンビエントの値を反映するにはパイプラインがアンビエント計算を行うようなモードになっていなければなりません。その設定は次のコードで行います。

```
pDevice->SetRenderState(D3DRS_AMBIENT,0x11111111);
```

第2引数の値は任意ですが、あまり大きな値にしないほうがいいでしょう（のっぺりしたレンダリングになってしまいます）。

結果は右の図のようになります。

▼実行結果



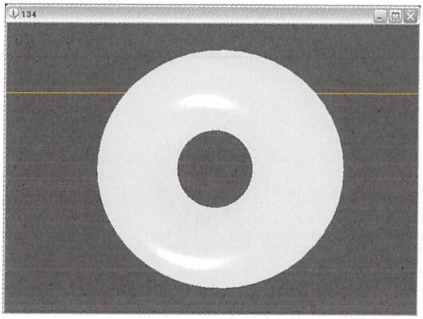
＊＊＊＊＊＊

Emissive（エミッシブ）は“自己発光”を意味するメンバです。

```
pMaterial->Emissive.r=0.5f;
pMaterial->Emissive.g=0.5f;
pMaterial->Emissive.b=0.5f;
```

結果は右の図のようになります。

▼実行結果



＊＊＊＊＊＊

コードをいちいち手作業書き換えなくても各メンバとその効果をインタラクティブに確認できるように、本サンプルはキー操作で各メンバを動的に変更できるようにしています。

対応するキーは次のとおりです。

●左右矢印キー

ディフューズの“R（赤）成分”のみ増減します。キーが多くなるので赤だけにしました。

●上下矢印キー

エミッシブをRGB成分均等に増減します。

●インサート・デリートキー

スペキュラーを増減します。

●ホーム・エンドキー

パワーを増減します。

One Point

ディフューズのa成分はアルファ値です。アルファ値は透明度を意味します。それについては、Tips138で解説します。

135　プラスチックのような質感にする

●アンビエントを非常に高くする

初級

サンプルプロジェクト　135　プラスチックのような質感

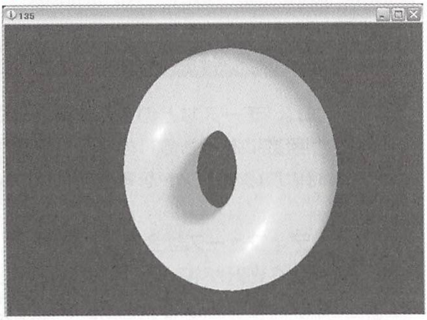
プラスティックは、微妙に表面付近が透けています。もちろんプラスティックと言っても様々ですが、それを考えていたらキリがありません。

当たった光のうちの少量が内部でも反射して表面が透けてみえることをトランスルーセントと呼びます（サブ・サーフェイス・スキャッティング　SSSとも言います）。「手の平を太陽に透かしてみーればー♪」っていう歌がありますよね、手の平は金属のように完全に光を遮断せず、僅かに光を透過させ、その結果、「まーっかーにーながーれるー僕の血潮ー♪」が見えるわけです。

トランスルーセントは固定機能シェーダーには無いので、実装するのであればプログラマブルシェーダーでということになりますが、ここではごく簡単にそれを近似してみたいと思います。

アンビエントのRGB成分のうち特定の成分を非常に高くすると、擬似トランスルーセントともいえるような効果が生まれます。

▼実行結果



```
    pMaterial->Ambient.r=1.0f;
    pMaterial->Ambient.g=10.0f;
    pMaterial->Ambient.b=1.0f;

    pMaterial->Diffuse.r=0.0f;
    pMaterial->Diffuse.g=0.8f;
    pMaterial->Diffuse.b=0.8f;
    pMaterial->Diffuse.a=1.0f;

    pMaterial->Specular.r=1.0f;
    pMaterial->Specular.g=1.0f;
    pMaterial->Specular.b=1.0f;
    pMaterial->Power=66.0f;

    pMaterial->Emissive.r=0.1f;
    pMaterial->Emissive.g=0.1f;
    pMaterial->Emissive.b=0.1f;
```

One Point

通常、ARGB成分の各成分の取りうる範囲は、0.0から1.0ですが、1.0以上に設定することもできます。ただし、1.0以上の値を反映するかどうかはビデオカード次第のようです。

136　ゴムのような質感にする

多孔物質のマテリアルを適用する

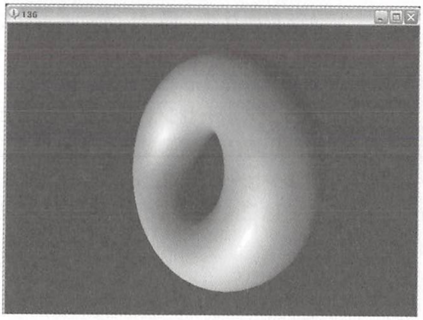
初級

サンプルプロジェクト　136　ゴムのような質感

ゴム、チーズなどの多孔物質（ミクロのあなが無数にあるもの）のマテリアルを作成する時は次のポイントを考慮すればいいでしょう。

1. ディフューズは金属より高い。
2. 全く光沢がないわけではないが、ごくわずか。そして金属のようにクッキリしたハイライトが出ない。

▼実行結果



```
pMaterial->Ambient.r=0.8f;
pMaterial->Ambient.g=0.8f;
pMaterial->Ambient.b=0.8f;

pMaterial->Diffuse.r=0.8f;
pMaterial->Diffuse.g=0.8f;
pMaterial->Diffuse.b=0.8f;
pMaterial->Diffuse.a=1.0f;

pMaterial->Specular.r=0.25f;
pMaterial->Specular.g=0.25f;
pMaterial->Specular.b=0.25f;
pMaterial->Power=16;

pMaterial->Emissive.r=0.0f;
pMaterial->Emissive.g=0.0f;
pMaterial->Emissive.b=0.0f;
```

1．によりディフューズは0.8
2．によりパワーは16とかなり低くし、テカリを広範囲に伸ばし、やわらかな質感を出します。

One Point

パワーは、柔らかい物質（ゴム）の時は低く、逆に硬い物質（金属）の時は高くします。

したがって、同じ金属でも鉄はアルミよりも高くします。

137 金属のような質感にする

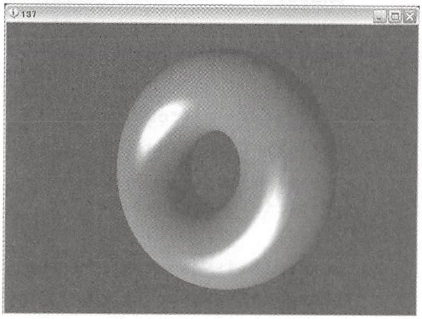
●固定機能シェーダーとプログラマブルシェーダーで表現する　上級

サンプルプロジェクト　137　金属のような質感
応用サンプルプロジェクト　137-2　金属のような質感（Pシェーダー）

●マテリアルオブジェクト（を固定機能シェーダーで反映する場合）

ここでは、金属のような質感を出すための2つのアプローチを解説します。一つは固定機能シェーダー、もう一つはプログラマブルシェーダーによるものです。

▼実行結果

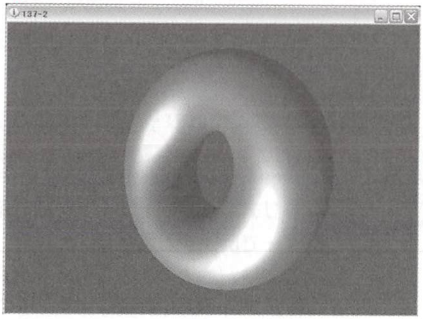


```
pMaterial->Ambient.r=0.5f;
pMaterial->Ambient.g=0.5f;
pMaterial->Ambient.b=0.5f;

pMaterial->Diffuse.r=0.5f;
pMaterial->Diffuse.g=0.5f;
pMaterial->Diffuse.b=0.5f;
pMaterial->Diffuse.a=1.0f;

pMaterial->Specular.r=2.0f;
pMaterial->Specular.g=2.0f;
pMaterial->Specular.b=2.0f;
pMaterial->Power=50;

pMaterial->Emissive.r=0.0f;
pMaterial->Emissive.g=0.0f;
pMaterial->Emissive.b=0.0f;
```

●プログラマブルシェーダーで、ピクセル単位にシェーディングする

マテリアルオブジェクトは必須ではありません。つまり、マテリアルオブジェクトがなくても、シェーダー側にディフューズ等のパラメーターを渡せば済みます。ユーザーが自由に決めた方法で“マテリアル自体”は設定しますが、固定機能シェーダーのように“マテリアルオブジェクト(D3DMATERIAL9)”は不要ということです。

▼実行結果



One Point

金属の場合、"周囲の景色を反射" させるとより金属らしさが出ます。反射については、Tips167～171で解説しています。

138 半透明にする

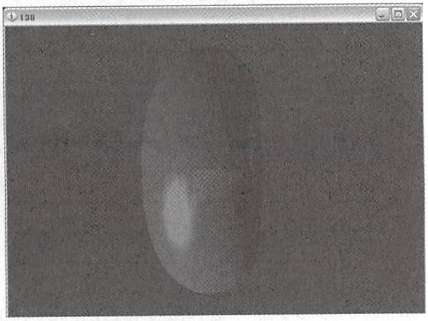
●ディフューズのアルファ成分を1.0以下にする

初級

サンプルプロジェクト　138 半透明にする

半透明にする際、マテリアル側の設定としてはディフューズのアルファ成分を1.0以下にすればいいだけです。

▼実行結果



```
pMaterial->Diffuse.r=0.5f;
pMaterial->Diffuse.g=0.5f;
pMaterial->Diffuse.b=0.5f;
pMaterial->Diffuse.a=0.5f;
```

ただ、ディフューズのアルファ成分の変更を反映するにはパイプラインが半透明計算を行うようなモードになっていなければなりません。その設定は次のコードで行います。

```
pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
pDevice->SetRenderState ( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
pDevice->SetRenderState ( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
```

One Point

ディフューズ以外のメンバ、例えばスペキュラーやエミッシブも同じD3DCOLORVALUE型であるためアルファ成分（a成分）を持ちますが、意味はなくそれらのアルファ成分を変更してもなんら変化はありません。

139 テクスチャとは

●本来は質感という意味

初級

テクスチャ（Texture）とは、本来は質感という意味です。質感という言葉の意味は広く、マテリアルも質感を表す概念です。

同じ質感を作り出すものでもマテリアルとは異なり、3DCGにおいてテクスチャとは、写真や模様の画像ファイルから取得した絵（ピクセル）を意味します。

時には画像ファイルそのものをテクスチャと呼ぶこともあります。これは間違いではありませんが、画像ファイル＝テクスチャではありません。

厳密に言うと「テクスチャになる画像ファイル」と呼ぶべきもので、省略した表現であることを頭に留めておかないと、本質を見失い、用語で混乱する原因になります。

140 テクスチャの作成と初期化

●D3DXCreateTextureFromFile関数、D3DXCreateTextureFromFileEx関数を使う

初級

テクスチャを作成する方法はいくつかあります。画像ファイルからではなく、絵を持たない空（から）のテクスチャオブジェクトを作成することもありますが、一般的には画像ファイルからテクスチャを作成することが多いので、ここではD3DXCreateTextureFromFile関数と、その拡張版であるD3DXCreateTextureFromFileEx関数を解説します。

●D3DXCreateTextureFromFile

D3DXCreateTextureFromFile関数はいたって簡単に画像からテクスチャを作成してくれます。

引数は3つしかありません。

第1引数　IDirect3DDevice9インターフェイスのポインター

第2引数　画像ファイル名

第3引数　IDirect3Dtexture9インターフェイスのポインターのアドレス

例）

```
IDirect3Dtexture9  *pTexture1=NULL;

D3DXCreateTextureFromFile(pDevice,"Tex1.jpg",&pTexture1);
```

●D3DXCreateTextureFromFileEx

この関数はD3DXCreateTextureFromFileの拡張関数であり、引数はいっきに多くなります。

第1引数　IDirect3DDevice9インターフェイスのポインター

第2引数　ファイル名

第3引数　画像の横幅

ゼロを指定すると、ファイルから取得されます。

第4引数　画像の縦幅

ゼロを指定すると、ファイルから取得されます。

第5引数　ミップマップレベルの数。

通常はNULLかD3DX_DEFAULTを指定して、自動的にミップマップを作成させます。ミップマップについてはTips142で解説しています。注意することは、ゼロにしたからといってミップマップの数がゼロになるということではなく、ゼロにすると“自動的にミップマップが作成される”ということであり、むしろ不要なミップマップを作成してメモリを余分に食うことになることが多いです。

最低のミップマップ数にしたいのなら、“1”を指定してください。

第6引数　レンダーターゲットあるいは動的テクスチャの指定

どちらも指定しない場合は“0”でも構いません。

第7引数、　テクスチャのフォーマットを指定します。

D3DFMT_UNKNOWNを指定すると画像ファイルのフォーマットでテクスチャを作成します。

第8引数　（メモリ上の）どこにテクスチャデータを格納するか

プールについては、Tips282、283を参照してください。

第9引数　フィルターの指定

通常はD3DX_DEFAULTで十分です。

第10引数　ミップマップのフィルターの指定

これも通常はD3DX_DEFAULTで十分です。

第11引数　抜け色の指定です。色は常に32ビットARGB値で表現します。

通常は“黒”を抜け色にする場合が多く、その場合0xff000000となります。抜け色を“青”にするなら0xff0000ffとなります。

第12引数　通常はNULLとします。

第13引数　通常はNULLとします。

第14引数　IDirect3Dtexture9インターフェイスのポインターのアドレス

例）200x100の画像ファイルを読み込むとして

```
IDirect3Dtexture9 *pTexture1=NULL;

D3DXCreateTextureFromFileEx(pDevice,"Tex1.jpg",200,100,0,0,D3DFMT_UNKNOWN,D3
DPOOL_DEFAULT,D3DX_FILTER_NONE,D3DX_DEFAULT,0xff000000,NULL,NULL,&pTexture1)
;
```

141 テクスチャのフォーマットとメモリサイズを理解する

●理論値と実際値の違いの理由を併せて理解する

中級

サンプルプロジェクト　141　テクスチャのメモリサイズ

テクスチャのサイズと言った場合、2つの意味があります。

1つは、テクスチャのピクセルサイズ

2つ目は、テクスチャのメモリ的なサイズ。言い換えるとテクスチャが占めるメモリ容量です。

ここでは、2つ目の意味であるメモリサイズを考えます。

●目安としての計算

テクスチャが占めるメモリ容量Spは、次の計算によりある程度予測できます。

Sp=横幅×縦幅×(ビット深度/8)(バイト)

例えば、256×256で24ビットカラーのテクスチャは、256×256×(24/8)=256×256×3=192キロバイト。また、400×300、16ビットカラーであれば400×300×(16/8)=400×300×2=234キロバイト、となります。

しかし、実際に作成されるDirect3Dテクスチャのメモリサイズは異なる場合があります。

その理由は、

1．Direct3Dがキャッシュを含む容量を確保する場合がある。その場合横幅は増加する

2．サポートされていないフォーマットの場合、上位のフォーマットでテクスチャが作成される。例えば古い8ビットフォーマットで作成しようとしても、ビデオカードが8ビットをサポートしていないため実際は16ビットフォーマットで作成されるなど。

3．ハードウェアの性能上の都合により特定のサイズに補正されてしまう場合。たとえば2の累乗でなければならない、4の倍数でなければならない、正方形でなければならないなど。

これらのどれもが、個々のハードウェア（ビデオカード）に依存する要因なので、どの原因がどのように現われるかは千差万別であり、サイズがどのように増加するかは予測不可能です。

●正確なメモリサイズ

前述の理由により、正確なメモリサイズはそれぞれの環境で異なり、それゆえにそれぞれの環境で測定するしかありません。

個々の環境での正確なメモリ容量Saは次の計算で求まります。

Sa＝ピッチ×縦幅（バイト）

まずピッチを求めます。ピッチとは、横幅と似ている概念です。横幅と同じ場合もあるし、それよりも大きな場合もあります。あるピクセルのメモリ上の位置から次のピクセルまでのバイト単位の距離です。言い換えると1つのピクセルに必要なバイト数です。これをストライドとも呼びます。

例えば、16ビットのテクスチャでは、あ

るピクセルから次のピクセルまでのバイト距離は、2バイト（16ビット）以上になります。画像の横幅とピッチが同じになる可能性があるのは、8ビットのテクスチャです。

　ただ、ピッチは複数の要因で、横幅よりも増加する可能性があります。

ピッチ＝横幅×ビット深度＋キャッシュ＋（そのフォーマットをサポートしていない場合の増加分）

　ピッチを取得するコードは、例えば次のようになります。なお、pTexはテクスチャオブジェクトポインターLPDIRECT3DTEXTURE9であるものとします。

```
D3DLOCKED_RECT Rect;
if(SUCCEEDED(pTex->LockRect(0,&Rect,NULL,D3DLOCK_READONLY)))
{
    Pitch=Rect=Rect.Pitch;

    pTex->UnlockRect(0);
}
```

　次に縦サイズを求めます。ここで注意しなくてはならないのは、画像の縦サイズではなく、テクスチャから取得した縦サイズを使用するということです。なぜなら、ビデオカードの都合により縦サイズも補正（変更）されているかもしれないからです。

　縦幅を取得するコードは、例えば次のようになります。

```
D3DSURFACE_DESC  Desc;
pTex->GetLevelDesc(0,&Desc);
RealHeight=Desc.Height
```

　テクスチャサイズは、Pitch　x　RealHeightです。

One Point

　テクスチャとして読み込む画像がjpeg、pngなどの圧縮画像だからといってテクスチャが占める容量が減ることはありません。結局テクスチャはビットマップ形式でのみピクセルを保持するので、最終的な容量は全て同じになります。

　圧縮画像は当然ビットマップ画像よりも汚いですが、それがそのままビットマップ化されるので、いい面がありません。圧縮タイプの画像ファイルのメリットと言えば、読み込み前のファイルサイズが小さいことくらいです。

142 ミップマップとメモリサイズを理解する

ミップマップの意味とメモリ容量に与える影響を理解する

中級

ミップマップの意味と機能

ミップマップとは、本来の画像をもとにして、その低解像度版をいくつか作成することにより、テクスチャのレンダリング速度を上げるテクニックです。

テクスチャがカメラから非常に遠くにあり、ほとんど米粒のように見えるような状況で、言い換えると米粒のようにレンダリングしてもいいような状況で、本来の解像度でレンダリングする必要はありません。その場合、低解像度のテクスチャを使用すればレンダリングにかかる時間は大幅に短縮できます。ミップマップは距離に応じて適当な解像度のテクスチャを選択しそれをレンダリングすることにより無駄なレンダリング時間を短縮します。

ミップマップとテクスチャ全体のメモリサイズ

低解像度版とは言え、本来の絵に加えて絵を複数枚用意するのですから、当然メモリサイズは増加します。

具体的には、横と縦のサイズが2分の1を画像を作成していきます。したがって、容量は直前の絵の4分の1になっていきます。

例えば、64X64の画像におけるミップマップは次のようになります。まず、本来の解像度で最上位のサーフェイスが作成されます。

64×64

次のサーフェイスは、縦横のサイズが半分なので

32×32

以降同様に、

16×16
8×8
4×4
2×2
1×1

となります。ビデオカードによっては最後は1×1ではなく2×2までとなるものもありますが、基本的にはこのようになります。

One Point

画像サイズが巨大な場合、例えば2000×2000ピクセルであったとしても、半分→半分…を繰り返すので、階層はそれほど多くはなりません。また、メモリもそれほど食わず、たとえミップレベルが無限（！）にあったとしてもミップマップが1枚のサイズと比べても必ず1.33倍以下に収まります（Σ計算ができる人は実際に計算すれば分かります）。

143 ミップマップの深さを変更する

●ミップマップの数を減らして(制限して)、メモリ容量を節約する　中級

参考サンプルプロジェクト　141　テクスチャのメモリサイズ

D3DXCreateTextureFromFileEx関数の第5引数で、ミップマップの枚数を指定できます。例えば、次のコードを実行するとミップマップは5枚作成されます。

```
D3DXCreateTextureFromFileEx(pDevice,"Tex2.bmp",0,0,5,0,D3DFMT_A8R8G8B8,D3DPO
OL_SYSTEMMEM,D3DX_FILTER_NONE,D3DX_DEFAULT,0xff000000,NULL,NULL,&pTex[Amount
])
```

ミップマップを1枚のみ（本来の解像度のテクスチャ1枚のみ）にしたい場合は、第5引数を1にします。第5引数をゼロにすると自動的にミップマップを作成することを意味してしまい、256ピクセルの画像の場合8枚程度のミップマップが作成されてしまうことに注意してください。

One Point

ミップマップを自動作成しても、サイズ的に元画像の1.33倍を越えることはありません。よほどメモリに余裕がない場合を除き、通常は自動で作成させるようにしたほうがいいのかもしれません。

144 アルファ付きテクスチャで透過部分を作る

●テクスチャアルファブレンドにより特定部分だけ透過させる　中級

サンプルプロジェクト　144　テクスチャの透過

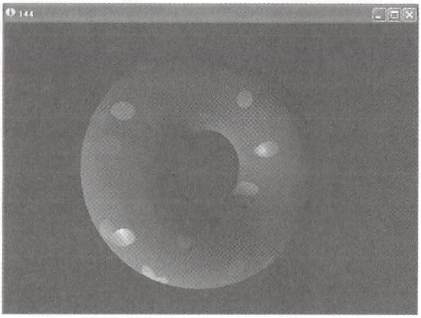
単純にシーン全体にアルファブレンドを適用（フレームバッファアルファ）すると、一様に透けてしまいますが、特定の部位だけ透けるようにしたい場合があります。それにはマテリアルごとに透明度を指定する方法（マテリアルアルファブレンド）とテクスチャにより透ける部位を指定する方法（テクスチャアルファブレンド）があります。ここではテクスチャアルファブレンドを行います。

テクスチャアルファブレンドではテクスチ

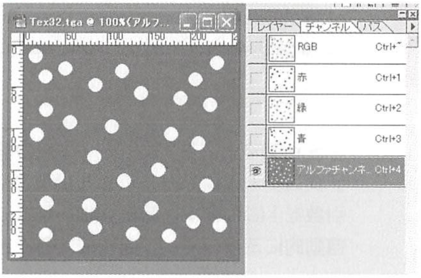
ャの画像自体にアルファ情報を埋め込みます。具体的には画像にアルファチャンネルを持たせます。

右下の図（144-2）はPhotoshop上で画像にアルファチャンネルを作成する様子です。一般的なビットマップ画像（24ビットビットマップ）ではなく、アルファチャンネルを持たせる画像フォーマットとしてはPNGとTGA（タルガ形式）などがあります。この例ではタルガ形式で作成しました。

▼実行結果（144-1）



▼アルファチャンネルを作成する様子（144-2）



丸い部分は不透明な部分です。それ以外の領域は透けます。完全な黒であれば100%透明になりますが、この例ではグレーなので100%透明ではなく「曇りガラス」のような透け方をします。図144-1を見ればその透け方が分かるでしょう。丸い部分は不透明になっています。

テクスチャアルファブレンドと言っても、コード側の設定（書き方）はフレームバッファアルファと同一で次のようにすればいいだけです。

```
//アルファブレンドを有効にする
    pDevice->SetRenderState(  D3DRS_ALPHABLENDENABLE,  TRUE  );
    pDevice->SetRenderState(  D3DRS_SRCBLEND,  D3DBLEND_SRCALPHA  );
    pDevice->SetRenderState(  D3DRS_DESTBLEND,  D3DBLEND_INVSRCALPHA  );
```

テクスチャアルファブレンドとして機能させるファクターは、テクスチャ画像の作成局面にあります。

One Point

お菓子の袋で、窓（中身が見える部位）を表現するのに有効でしょう。アルファチャンネルで窓の部分をその他の部位より相対的に暗めにすれば実現できます。

145 マルチパスとは

複数回のレンダリングのこと

初級

参考サンプルプロジェクト　155　マルチパステクスチャブレンディング

ここで言う「パス」とは、「一回のレンダリング」のことです。したがってマルチパスとは、複数回のレンダリングのことです。通常、プリミティブやメッシュをレンダリングする場合、パスは1回です（1回で十分ですよね）。

マルチパスにより何回も同じジオメトリをレンダリングするのは時間の無駄ではありますが、それによりテクスチャをブレンドすることができます。

最初のパスでは、デカールテクスチャを貼ったジオメトリをレンダリングし、2回目のレンダリングでは“汚れテクスチャ”を貼った（同じ）ジオメトリをレンダリングすることで、あたかも2枚のテクスチャがブレンドされているかのような効果を出します。

擬似コードで書くと次のようになります。

```
//2つのパスのブレンドの仕方を指定する
SetTextureStageState( 0, D3DTSS_COLOROP,    D3DTOP_MODULATE );
SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_CURRENT );

//1回目のレンダリング
SetTexture(0,デカールテクスチャ)
DrawSubset() や  DrawPrimitive()

//2回目のレンダリング
SetTexture(0,汚れのテクスチャ)
DrawSubset() や  DrawPrimitive()
```

SetTextureに指定するステージが2つとも0（同じテクスチャステージ）であることに注意してください。2階のレンダリングを行うのでステージが同じであってもいいわけです。逆に言うと、異なるステージを処理できないカードのための手法とも言えます。テクスチャステージについては、Tips147で解説しています。

One Point

多くのグラフィックカードは、シングルパスでのテクスチャブレンディングがサポートされているので、マルチパスを使用する局面は少ないかもしれません。

146 マルチテクスチャとは

●複数のテクスチャを重ねて貼ること

初級

ここで言うマルチテクスチャとは正確に言うと「マルチプルテクスチャブレンディング」のことです。長いのでマルチテクスチャと言われています。なんのことかと言うと、要するに「複数のテクスチャを重ねて貼ること」をマルチテクスチャと言っているわけです。

マルチテクスチャにはレンダリングを1回で実現するか複数回しなければならいかというレンダリング回数を基準にすると2通りのアプローチがあります。

- マルチパスマルチテクスチャ（ブレンディング）
- シングルパスマルチテクスチャ（ブレンディング）

シングルパス～のほうは、1回のレンダリングで済みます。つまり、レンダリングされる前にすでにテクスチャがブレンドされていて、そのブレンドされたテクスチャを貼ればいいわけなので1回で済みます。

一方、マルチパス～のほうは、（主に）テクスチャレベルでブレンドできないカードの場合に採る手法で、個々の（ブレンドされていない）テクスチャを貼ったジオメトリをまずレンダリングします。それをテクスチャを変えて同じことを繰り返し、レンダリングを複数回行なわなければなりません。このことはTips145でも解説しています。

One Point

もちろんどちらの手法でも、複数のテクスチャが重ねて貼るという結果は同じです。現在ほとんどのビデオカードでは、シングルパスでのマルチテクスチャをサポートしています。

147 テクスチャステージとは

●シングルパスにおけるレイヤーのこと

初級

テクスチャステージとはシングルパスにおけるレイヤーのことです。画像編集ソフトのPhotoshopおけるレイヤーを想像すると分かりやすいと思います。レイヤー毎に1枚ずつテクスチャを持つことができ、たとえばステートを3つ使うとうことは、言い換えればレイヤーを3枚使うということであり、異なるテクスチャを3枚ブレンドすることができます。

シングルパスでステージを使えないカードの場合は、パス自体を複数繰り替えさなくてはなりなせん。それはマルチパスといいます。マルチパスについてはTips145を参照してください。

ステージとはあくまでもシングルパスにおけるレイヤーのことであると覚えましょう。

One Point

今まで何気なくSetTexture(0,~)と書いていたのは、ステート0番にテクスチャをセットしていたのです。テクスチャブレンドしない場合のステートはゼロ番にセットします。

148 ブレンドできるテクスチャの数を調べる

グラフィックカードがテクスチャブレンドをサポートしているかどうか確認する　初級

サンプルプロジェクト　148　ブレンドできるテクスチャの数

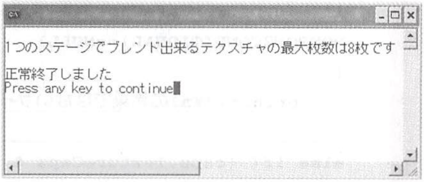
Direct3Dでは最高8枚までテクスチャをブレンドすることができます。そして、現在のグラフィックカードはほとんど8枚までブレンドすることができます。

念のため、グラフィックカードが本当に8枚まで対応しているかどうか、調べることができます。

サンプルは、単にDirect3Dを初期化し、グラフィックカードの能力を取得するだけなので、コンソールアプリケーションとして作成しました。

ポイントとなる部分は次のたった3行のコード片です。

▼実行結果



```
1つのステージでブレンド出来るテクスチャの最大枚数は8枚です

正常終了しました
Press any key to continue
```

```
//デバイス能力を調査
    D3DCAPS9 caps;
    pDevice->GetDeviceCaps( &caps );
    printf("¥n1つのステージでブレンドできるテクスチャの最大枚数は%d枚です¥n",
caps.MaxTextureBlendStages);
```

当然ですが、本Tipsは同時にグラフィックカードがシングルパスマルチテクスチャをサポートしているかどうか確認することでもあります。

One Point

本サンプルのように、グラフィック表示が不要な場合は、コンソールウィンドウ上でDirect3Dを初期化すると便利です。

149 2の累乗サイズしかサポートしていないか調べる

●グラフィックカードが2の累乗以外のサイズでも表示可能かどうか確認する　初級

サンプルプロジェクト　149　2の累乗サイズ

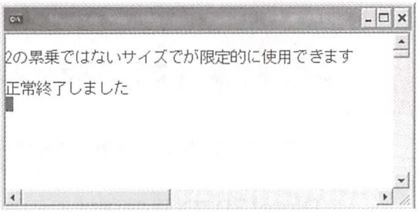
2の累乗とは、2乗(4)、3乗(8)、4乗(16)…8乗(256)、9乗(512)、10乗(1024)…（以降同様)のことです。

カードによっては、これらのサイズしかサポートしていないものがあるようです。そうようなカードでは例えば、

100×100ピクセルというサイズのテクスチャが扱えないことになります。

本サンプルはグラフィックカードが2の累乗以外のサイズをサポートしているかどうかを表示します。

▼実行結果



```
2の累乗ではないサイズでが限定的に使用できます
正常終了しました
```

```
D3DCAPS9 caps;
pDevice->GetDeviceCaps( &caps );
if( !caps.TextureCaps && (caps.TextureCaps & D3DPTEXTURECAPS_
NONPOW2CONDITIONAL)==NULL)
{
    printf("¥n2の累乗ではないサイズは無条件で使用できます¥n");
}
else if( (caps.TextureCaps & D3DPTEXTURECAPS_NONPOW2CONDITIONAL)=
=D3DPTEXTURECAPS_NONPOW2CONDITIONAL)
{
    printf("¥n2の累乗ではないサイズでが限定的に使用できます¥n");
}
```

One Point

本サンプルのように、グラフィック表示が不要な場合は、コンソールウィンドウ上でDirect3Dを初期化すると便利です。

150 正方形しかサポートしていないか調べる

●グラフィックカードが正方形以外のテクスチャをサポートしているかどうか調べる　初級

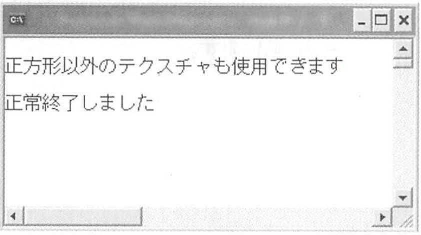
サンプルプロジェクト　150　正方形以外のサポート

グラフィックカードによっては、テクスチャが正方形でなければならないものがあるようです（私はそうのようなカードを知りませんが…）。

正方形とは、つまり横サイズと縦サイズが同じでなければならないということで、例えば横が256ピクセルであれば縦も256でなければならず、256X512など、長方形は表示できないということになります。

念のため調べる場合のコードは次のようになります。

▼実行結果



```
正方形以外のテクスチャも使用できます
正常終了しました
```

```
D3DCAPS9 caps;
pDevice->GetDeviceCaps( &caps );
if(( caps.TextureCaps & D3DPTEXTURECAPS_SQUAREONLY)==D3DPTEXTURECAPS_
SQUAREONLY)
{
    printf("¥nテクスチャのピクセルサイズは正方形しかサポートしていません¥n");
}
else
{
    printf("¥n正方形以外のテクスチャも使用できます¥n");
}
```

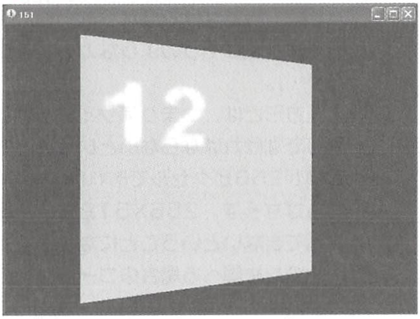
151 プリミティブに2枚のテクスチャをブレンドして貼る

●シングルパスで2枚のテクスチャをブレンドする　中級

サンプルプロジェクト　151　プリミティブに2枚をブレンド

シングルパスでテクスチャをブレンドする場合、レンダリングの前にテクスチャレベル（テクスチャ自体）でブレンドするわけですが、それぞれのテクスチャのレイヤーはステージと言います。

▼実行結果



次のように、それぞれのステージにおいて、どのようにブレンドするかを設定します。

```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //ステージ0
    pDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,   D3DTOP_MODULATE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
    // ステージ1
    pDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
    //テクスチャをセット
    pDevice->SetTexture(0,pTexture1);
    pDevice->SetTexture(1,pTexture2);
    //レンダリング
    pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX2);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof(TEX_
VERTEX));

    pDevice->EndScene();
}
```

最初のステージ（ステージ0）では、（ステージ0の）テクスチャとプリミティブのディフューズをブレンドし、

```
pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
```

次のステージ1では、その時のテクスチャ（つまりディフューズとステージ0のテクスチャのブレンド色）にステージ1のテクスチャをブレンドします。

```
pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
```

なお、pDevice->SetTextureStageState(　0,　D3DTSS_TEXCOORDINDEX,　0　);の意味は、どのテクスチャ座標を使用するかということです。本サンプルではテクスチャ座標は1つなのであまり意味はありません。テクスチャ座標が複数ある場合には（TEX2フォーマット等）有効です。

One Point

ステージのインデックスとステージ数は1つズレることに留意してください（インデックスはゼロから始まるので）。

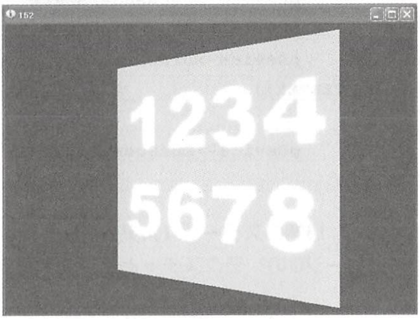
152　プリミティブに8枚のテクスチャをブレンドして貼る

●シングルパスで最大枚数（8枚）のテクスチャをブレンドする　中級

サンプルプロジェクト　152　プリミティブに8枚をブレンド

手法はTips151と全く同じです。異なるのは、テクスチャの数が増えているだけのことです。コードは掲載しなくてもいいかと思いますが、念のため載せておきます。

▼実行結果



```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //ステージ0
    pDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,    D3DTOP_MODULATE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
    // ステージ1
    pDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ2
    pDevice->SetSamplerState( 2, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 2, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 2, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 2, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 2, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 2, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ3
    pDevice->SetSamplerState( 3, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 3, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 3, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 3, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 3, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 3, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ4
    pDevice->SetSamplerState( 4, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 4, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 4, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 4, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 4, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 4, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ5
    pDevice->SetSamplerState( 5, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 5, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 5, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 5, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 5, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 5, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ6
    pDevice->SetSamplerState( 6, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 6, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 6, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 6, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 6, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 6, D3DTSS_COLORARG2, D3DTA_CURRENT );
    // ステージ7
    pDevice->SetSamplerState( 7, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 7, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 7, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 7, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 7, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 7, D3DTSS_COLORARG2, D3DTA_CURRENT );
    //テクスチャをセット
    pDevice->SetTexture(0,pTexture1);
    pDevice->SetTexture(1,pTexture2);
    pDevice->SetTexture(2,pTexture3);
    pDevice->SetTexture(3,pTexture4);
    pDevice->SetTexture(4,pTexture5);
    pDevice->SetTexture(5,pTexture6);
    pDevice->SetTexture(6,pTexture7);
    pDevice->SetTexture(7,pTexture8);
    //レンダリング
    pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof(TEX_VERTEX));

    pDevice->EndScene();
}
```

One Point

2枚目以降のブレンド引数設定は全て、そのステージのテクスチャD3DTA_TEXTUREと現在のブレンド色D3DTA_CURRENTを冗長に繰り返しているだけです。

153 メッシュに2枚のテクスチャをブレンドして貼る

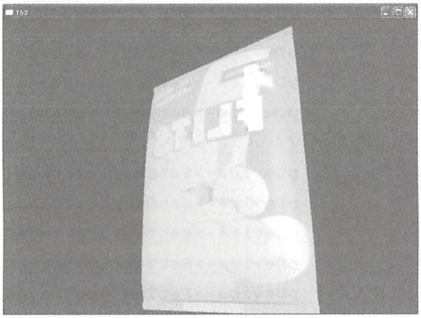
シングルパスで2枚のテクスチャをブレンドする

中級

サンプルプロジェクト　153 メッシュに2枚をブレンド

手法はTips151と全く同じです。

▼実行結果



```
if( SUCCEEDED( pDevice->BeginScene() ) )
 {
     //ステージ0
    pDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,    D3DTOP_MODULATE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );

    // ステージ1
    pDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
    pDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
    pDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
    pDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
    pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
```

```
   pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
   //テクスチャをセット
   pDevice->SetTexture(1,pTexture1);

       for( DWORD i=0; i<dwNumMaterials; i++ )
      {
           pDevice->SetMaterial( &pMeshMaterials[i] );
           pDevice->SetTexture( 0, pMeshTextures[i] );
           pMesh->DrawSubset( i );
       }
    pDevice->EndScene();
}
```

One Point

テクスチャをブレンドするとそれぞれの色成分が合成されるので、このサンプルではポテトチップスの袋が緑色っぽくなります。

154 メッシュに8枚のテクスチャをブレンドして貼る

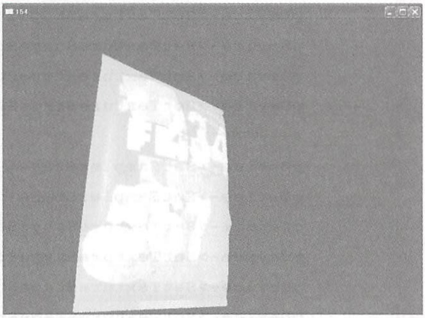
●シングルパスで最大枚数（8枚）のテクスチャをブレンドする

中級

サンプルプロジェクト　154 メッシュに8枚をブレンド

手法はTips153と全く同じです。異なるのは、テクスチャの数が増えているだけのことです。

▼実行結果



```
if( SUCCEEDED( pDevice->BeginScene() ) )
 {
    //ステージ0
```

```
        pDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,    D3DTOP_MODULATE );
        pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
        // ステージ1
        pDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
        pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
        // ステージ2
        pDevice->SetSamplerState( 2, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 2, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 2, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 2, D3DTSS_COLOROP, D3DTOP_ADD );
        pDevice->SetTextureStageState( 2, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 2, D3DTSS_COLORARG2, D3DTA_CURRENT );
        // ステージ3
        pDevice->SetSamplerState( 3, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 3, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 3, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 3, D3DTSS_COLOROP, D3DTOP_ADD );
        pDevice->SetTextureStageState( 3, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 3, D3DTSS_COLORARG2, D3DTA_CURRENT );
        // ステージ4
        pDevice->SetSamplerState( 4, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 4, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 4, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 4, D3DTSS_COLOROP, D3DTOP_ADD );
        pDevice->SetTextureStageState( 4, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 4, D3DTSS_COLORARG2, D3DTA_CURRENT );
        // ステージ5
        pDevice->SetSamplerState( 5, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 5, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
        pDevice->SetTextureStageState( 5, D3DTSS_TEXCOORDINDEX, 0 );
        pDevice->SetTextureStageState( 5, D3DTSS_COLOROP, D3DTOP_ADD );
        pDevice->SetTextureStageState( 5, D3DTSS_COLORARG1, D3DTA_TEXTURE );
        pDevice->SetTextureStageState( 5, D3DTSS_COLORARG2, D3DTA_CURRENT );
        // ステージ6
        pDevice->SetSamplerState( 6, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
        pDevice->SetSamplerState( 6, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
```

```
  pDevice->SetTextureStageState( 6, D3DTSS_TEXCOORDINDEX, 0 );
  pDevice->SetTextureStageState( 6, D3DTSS_COLOROP, D3DTOP_ADD );
  pDevice->SetTextureStageState( 6, D3DTSS_COLORARG1, D3DTA_TEXTURE );
  pDevice->SetTextureStageState( 6, D3DTSS_COLORARG2, D3DTA_CURRENT );
  // ステージ7
  pDevice->SetSamplerState( 7, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
  pDevice->SetSamplerState( 7, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
  pDevice->SetTextureStageState( 7, D3DTSS_TEXCOORDINDEX, 0 );
  pDevice->SetTextureStageState( 7, D3DTSS_COLOROP, D3DTOP_ADD );
  pDevice->SetTextureStageState( 7, D3DTSS_COLORARG1, D3DTA_TEXTURE );
  pDevice->SetTextureStageState( 7, D3DTSS_COLORARG2, D3DTA_CURRENT );
  //テクスチャをセット
  pDevice->SetTexture(1,pTexture1);
  pDevice->SetTexture(2,pTexture2);
  pDevice->SetTexture(3,pTexture3);
  pDevice->SetTexture(4,pTexture4);
  pDevice->SetTexture(5,pTexture5);
  pDevice->SetTexture(6,pTexture6);
  pDevice->SetTexture(7,pTexture7);

    for( DWORD i=0; i<dwNumMaterials; i++ )
       {
           pDevice->SetMaterial( &pMeshMaterials[i] );
           pDevice->SetTexture( 0, pMeshTextures[i] );
           pMesh->DrawSubset( i );
       }
    pDevice->EndScene();
}
```

One Point

2枚目以降のブレンド引数設定は全て、そのステージのテクスチャD3DTA_TEXTUREと現在のブレンド色D3DTA_CURRENTを冗長に繰り返しているだけです。

155 マルチパスでテクスチャをブレンドする

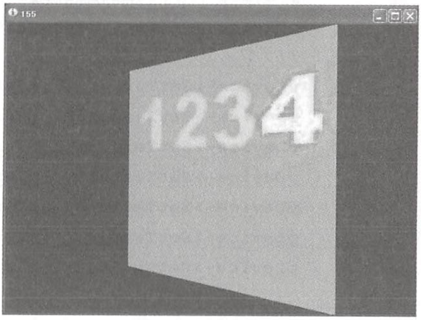
●レンダリングを複数回行うことにより結果的にテクスチャを重ねる効果を出す　中級

サンプルプロジェクト　155　マルチパステクスチャブレンディング

マルチパスでは、“レンダリングを複数回”行うことにより、テクスチャを重ねます。詳細についてはTips145を参照してください。

マルチパスでテクスチャを重ねるコードは次のようになります。この例では4回のレンダリングにより4枚のテクスチャを重ねています。

▼実行結果



```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);

    //ブレンド具合を設定する
    pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,   D3DTOP_MODULATE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_CURRENT );

    //パス1
    {
        //テクスチャをセット
        pDevice->SetTexture(0,pTexture1);
        //最初のテクスチャは単純にレンダリング
        pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof
(TEX_VERTEX));
    }
    //パス2  (2枚目以降からブレンド具合を設定する必要あり)
    {
        //テクスチャをセット
        pDevice->SetTexture(0,pTexture2);
        pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
        pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCCOLOR );
        pDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_INVSRCCOLOR );

        //レンダリング
        pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof
        (TEX_VERTEX));
    }
    //パス3
    {
        //テクスチャをセット
        pDevice->SetTexture(0,pTexture3);

        pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
        pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCCOLOR );
        pDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_INVSRCCOLOR );

        //レンダリング
        pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof
        (TEX_VERTEX));
    }
    //パス4
    {
        //テクスチャをセット
        pDevice->SetTexture(0,pTexture4);

        pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
        pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCCOLOR );
        pDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_INVSRCCOLOR );

        //レンダリング
        pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint, sizeof
        (TEX_VERTEX));
    }
    pDevice->EndScene();
```

One Point

マルチパスは、シングルパス（同じ枚数で、結果が同じものでも）に比べてパフォーマンスが落ちます。

156 テクスチャのないメッシュにテクスチャを貼る

●テクスチャ座標を持たないメッシュにテクスチャ座標を追加する

上級

サンプルプロジェクト　156 テクスチャの無いメッシュにテクスチャを貼る

テクスチャ座標を持たないメッシュ、例えばD3DX関数で自動生成できる基本形状メッシュはテクスチャ座標を持たず、また、自分で作成したXファイルメッシュもテクスチャを貼っていなければ、テクスチャ座標を持っていません。

そのようなメッシュに、後から（プログラム内で）テクスチャを貼り付けたい場合がありますが、どのようにすればできるのでしょう？

テクスチャ座標を追加すればいいのです。それから、追加したテクスチャ座標を意図したマッピング形態（プレーンマッピングなど）に初期化する必要もあります。

サンプルでは、D3DX関数で自動生成したトーラスメッシュに、後からテクスチャ座標を追加し、プレーンマッピングされるようにテクスチャ座標を初期化しています。

まず、トーラスメッシュを普通に作成します。

```
//トーラスの作成
LPD3DXMESH pTempMesh=NULL;
D3DXCreateTorus(pDevice,0.5,1,48,48,&pTempMesh,NULL);
```

当然この時点ではトーラスメッシュにテクスチャを貼り付けることはできません。

テクスチャ座標を持つような頂点フォーマットによりメッシュを変更複製します。

```
//一時的なトーラスメッシュから、テクスチャ座標を持つメッシュとして複製を作成する
pTempMesh->CloneMeshFVF(NULL,D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1,
pDevice,&pTorusMesh);
SAFE_RELEASE(pTempMesh);
```

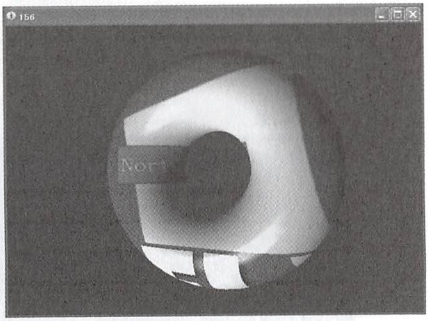
ここでは最も簡単なプレーンマッピングによりテクスチャ座標を計算しています。

```
//テクスチャ座標の初期化
LPDIRECT3DVERTEXBUFFER9 pVB=NULL;
TEX_VERTEX ＊pVertices=NULL;
pTorusMesh->GetVertexBuffer(&pVB);
DWORD NumVertices=pTorusMesh->GetNumVertices();
DWORD dwStride=pTorusMesh->GetNumBytesPerVertex();
D3DXVECTOR3 vMax,vMin;
FLOAT fWidth,fHeight;
if(SUCCEEDED(pVB->Lock(0,0,(VOID＊＊)&pVertices,0)))
{
    D3DXComputeBoundingBox( (D3DXVECTOR3＊)pVertices,NumVertices,dwStride,
    &vMax,&vMin);
    fWidth=fabs(vMax.x-vMin.x);
    fHeight=fabs(vMax.y-vMin.y);

    for(DWORD i=0;i<NumVertices;i++)
    {
            pVertices[i].u=vMin.x-(pVertices[i].Coord.x / fWidth);
            pVertices[i].v=vMin.y-(pVertices[i].Coord.y / fHeight);
    }
    pVB->Unlock();
}
```

計算したテクスチャ座標は、頂点バッファをロックし、実際に頂点バッファーに代入します。

これで、SetTextureを実行すればトーラスメッシュにはテクスチャが貼られます。

▼実行結果



157 テクスチャ座標を自動生成する

●IDirect3DDevice9::SetTextureStageState(n,D3DTSS_TEXCOORDINDEX,~)メソッドを使う 中級

サンプルプロジェクト 157 (特殊な)テクスチャ座標の自動生成

最初に述べておきますが、“テクスチャ座標の自動生成” と言ってもその用途は限定的です。通常は、テクスチャ座標は静的に作成したものをメッシュ自身が保持している必要があります。ここでのテクスチャ座標は特殊なものなので、それ故その用途は限定的です（キューブマッピングに利用したり、鏡のエフェクト用に使用したりします）。

コードはたった1行です。

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_CAMERA
SPACEPOSITION );
```

これだけです。このコードは次の図のようなテクスチャマッピングを行うテクスチャ座標を作成します。このテクスチャ座標は、ジオメトリの回転（や位置）に影響されないので、シーンのスナップショット画像をテクスチャ画像にすれば“擬似的な鏡”の効果が得られます。また、第3引数を変えると次のような効果に切り替わります。

▼157-2

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_CAMERA
SPACENORMAL );
```

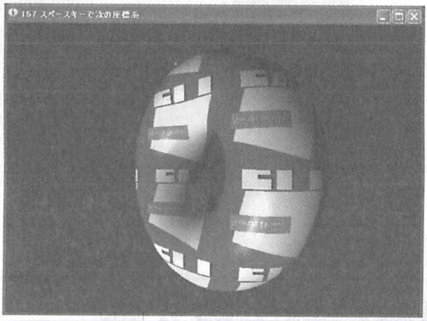
▼157-3

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_CAMERA
SPACEREFLECTIONVECTOR );
```

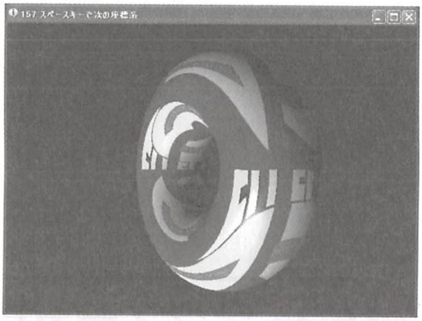
▼157-4

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_SPHEREMAP
);
```

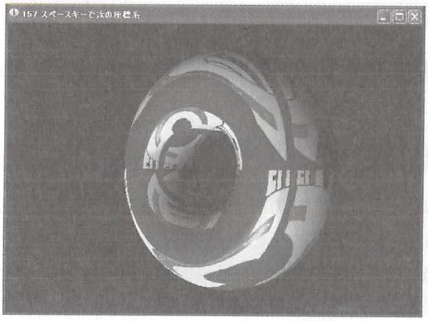
▼実行結果


157 スペースキーで次の座標系

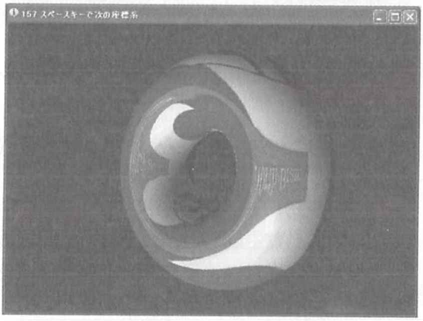
▼実行結果（157-2）


157 スペースキーで次の座標系

▼実行結果（157-3）


157 スペースキーで次の座標系

▼実行結果（157-4）


157 スペースキーで次の座標系

なお、これらのレンダリングステートを適用した以降にレンダリングされるメッシュは全てこの影響を受けるので、通常のテクスチャマッピングが行われません（通常のテクス

チャ座標に優先してこの特殊なテクスチャ座標が適用されてしまいます）。そのため、このエフェクトを無効にする必要があります。

無効にする場合は打3引数をパススルーにします。

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, D3DTSS_TCI_PASSTHRU
);
```

これにより、以降のテクスチャマッピングは通常通り行われます。

つまり、次のようにするわけです。

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_CAMERA
SPACEPOSITIONなど ); ←自動生成をオン
```

↓

（特殊なテクスチャ座標を適用したいジオメトリのレンダリング）

↓

```
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, D3DTSS_TCI_PASSTHRU
);←自動生成をオフ
```

↓

（通常のテクスチャ座標によるレンダリング）

158 テクスチャ自体を動かす

●IDirect3Ddevice9::SetTextureStageState(0, D3DTSS_TEXTURETRANSFORMFLAGS, XXX)を設定する

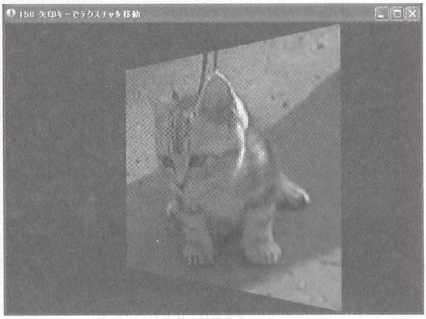
中級

サンプルプロジェクト　158 テクスチャ自身を動かす

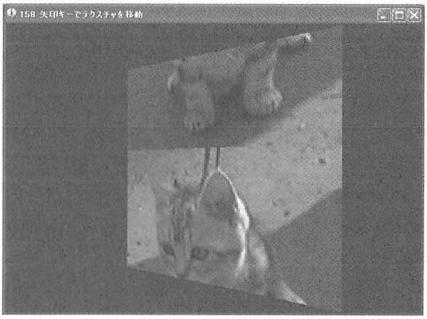
ジオメトリを動かすのではなく、ジオメトリ上に貼っているテクスチャそのものを動かすことができます。

サンプルでは、テクスチャの平行移動を行っています。

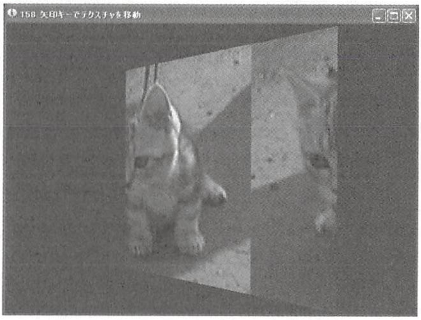
▼実行結果



▼実行結果



▼実行結果



```
//テクスチャ移動処理
{
    D3DXMATRIX mat;
    D3DXMatrixIdentity(&mat);
    mat._31=U;
    mat._32=V;
    pDevice->SetTransform(D3DTS_TEXTURE0, &mat);
    pDevice->SetTextureStageState(0, D3DTSS_TEXTURETRANSFORMFLAGS, D3DTTFF_
    COUNT2);
}
//テクスチャをセット
pDevice->SetTexture(0,pTexture);
```

平行移動は3行1列の要素にU値、3行2列の要素にV値を代入します。

なお、UV座標はグローバルに用意した変数をウィンドウプロシージャー内でキー入力により更新しています。

```
FLOAT  U=0,V=0;
```

```
case WM_KEYDOWN:
    switch(wParam)
    {
    case VK_LEFT:
        U+=0.01;
        break;
    case VK_RIGHT:
        U-=0.01;
        break;
    case VK_UP:
        V+=0.01;
        break;
    case VK_DOWN:
        V-=0.01;
        break;
    }
    break;
```

＊＊＊＊＊＊

ちなみに回転の場合は、回転角をθラジアンとすると、行列の11要素にcosθ、12要素にsinθ 21要素に-sinθ、22要素にcosθを代入します。

サンプルの変数Uを利用して回転させるとすると

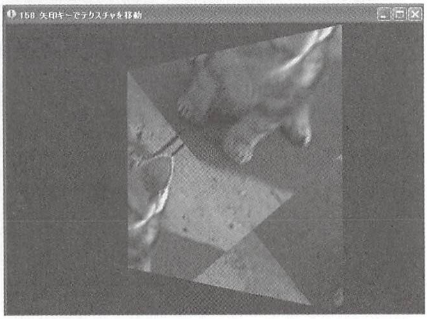
```
mat._31=U;
mat._32=V;
```

の部分を次のように変更します。

```
mat._11=cos(U);
mat._12=sin(U);
mat._21=-sin(U);
mat._22=cos(U);
```

結果は右の図のように、ちゃんとキー入力により回転します。

▼実行結果



＊＊＊＊＊＊

スケーリングの場合は、行列の11要素にX方向のスケール係数、22要素にY方向のスケール係数を代入します。

サンプルの変数Uを利用してスケーリングさせるとすると

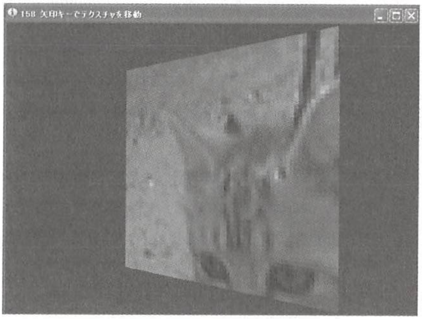
```
mat._31=U;
mat._32=V;
```

の部分を次のように変更します。

```
mat._11=U;
mat._22=U;
```

　結果は右の図のとおりです。矢印キーで適当に拡大縮小してみてください。

▼実行結果


158 矢印キーでテクスチャを移動

159 スプライトのテクスチャを反転させる

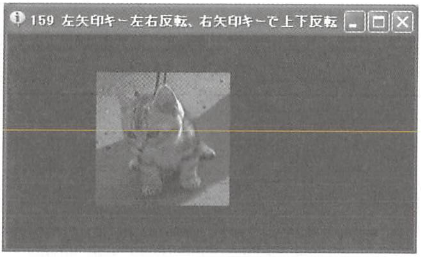
●スケーリング行列を適用する

中級

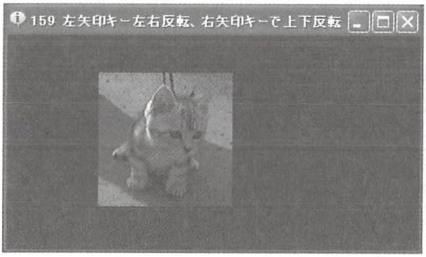
サンプルプロジェクト　159　スプライトのテクスチャを反転

　ここでは、スプライトそのものを動かすのではなく、スプライト内のテクスチャだけを動かすことについて解説します。

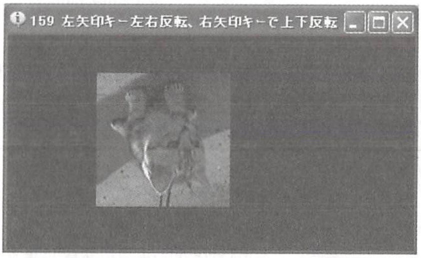
▼実行結果


159 左矢印キー－左右反転、右矢印キーで上下反転

▼実行結果


159 左矢印キー－左右反転、右矢印キーで上下反転

▼実行結果


159 左矢印キー－左右反転、右矢印キーで上下反転

コードは次のとおりです。

```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    RECT rect={0,0,SPRITE_WIDTH,SPRITE_HEIGHT};
    D3DXVECTOR3 vec3Center(SPRITE_WIDTH/2,SPRITE_HEIGHT/2,0);
    D3DXMATRIX matWorld,matScale,matTranslation;
    D3DXMatrixScaling(&matScale,g_ReverseH,g_ReverseV,1);
    D3DXMatrixTranslation(&matTranslation,g_vecPosition.x,g_vecPosition.y,
    0);
    matWorld=matScale * matTranslation;
    pSprite->SetTransform(&matWorld);

    pSprite->Begin(D3DXSPRITE_ALPHABLEND);
    pSprite->Draw(pTexture,&rect,&vec3Center,NULL,D3DCOLOR_ARGB(255,255,255,
    255));
    pSprite->End();
    pDevice->EndScene();
}
```

なお、スケーリングの基にしている変数は　変更しています。
ウィンドウプロシージャーでキー入力により

```
case WM_KEYDOWN:
switch((CHAR)wParam)
{
    case VK_ESCAPE:
        PostQuitMessage(0);
        break;
    case VK_LEFT:
        g_ReverseH=-g_ReverseH;
        break;
    case VK_RIGHT:
        g_ReverseV=-g_ReverseV;
        break;
}
```

One Point

g_ReverseHは左右の反転に用い、
g_ReverseVは上下の反転に用いています。

160 プログラマブルシェーダーでテクスチャを貼る

●プログラマブルシェーダーにおけるテクスチャの扱い方を知る

上級

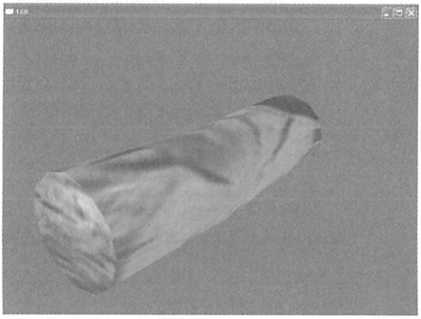
サンプルプロジェクト　160 Pシェーダーでテクスチャを貼る

ここでは、固定機能シェーダーではくプログラマブルシェーダーにおけるテクスチャのセットを行います。

結果としての絵は同じですが、プログラマブルシェーダーの場合は、より柔軟にテクスチャのピクセルを操作できるので便利です(利点については、Tips163、164にサンプルがあります)。

固定機能シェーダーにおけるテクスチャのセットは、IDirect3DDevice9::SetTextureによりテクスチャをセットしますが、プログラマブルシェーダーの場合は、当該シェーダーのインターフェイス内のID3DXEffect::SetTextureメソッドによりセットします。

▼実行結果



```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    g_pEffect->Begin( NULL, 0 );
    for( DWORD i=0; i<dwNumMaterials; i++ )
    {
        g_pEffect->SetTexture("DecalTexture",pMeshTextures[i]);
        g_pEffect->BeginPass(0);
        pMesh->DrawSubset( i );
        g_pEffect->EndPass();
    }
    g_pEffect->End();
    pDevice->EndScene();
}
```

そして、シェーダー側でのコードの書き方は次のようになります。

```
///////////////////////////
//グローバル
///////////////////////////
texture DecalTexture;

////////////////////////////
//定義
///////////////////////////
sampler DecalSampler = sampler_state
{
    Texture = (DecalTexture);
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = NONE;
    ddressU = Clamp;
    ddressV = Clamp;
};

///////////////////////////
//ピクセル・シェーダー
////////////////////////////
float4 PS( float2 TCoord : TEXCOORD  ) : COLOR
{
    float4 FinalColor=tex2D(DecalSampler ,TCoord);

    return FinalColor;
}
/////////////////////////
// テクニック
//////////////////////////
technique PShader
{
    pass p0
    {
            PixelShader = compile ps_2_0 PS();
    }
}
```

シェーダーではテクスチャとサンプラーが対で必要です。
基本的な考え方は、何（テクスチャ）をどのように（サンプラー）という考え方です。サンプラーはテクスチャの貼り方を表しています。

ここでは、一般的なデカールテクスチャのサンプルタイプを書きましたが、もちろん様々なサンプルが可能です。

なお、テクスチャを貼ることだけが目的なので、頂点シェーダーは用いず、最小限のシェーダーとしてコーディングしました。

One Point

テクスチャの設定（SetTexture）は、BeginPassとEndPassの間に入れないようにしましょう。本サンプルではテクスチャが1枚なので上手くいきますが、複数ある場合にはおかしなことになります。BeginPassとEndPassの間には純粋にレンダリングだけを行ってください（DrawSubsetやDrawPrimitive）。

161 マルチテクスチャでメッシュを「燃やす」

●一定の間隔で燃焼部分のテクスチャをブレンドすることによって、木炭の燃焼を表現してみる

上級

サンプルプロジェクト　161　テクスチャで燃やす

技術的には、Tips153の「メッシュにテクスチャをブレンドする」と同じです。

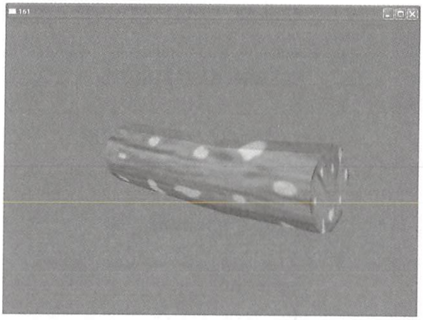
木炭そのもののテクスチャと、木炭が燃えて赤くなっている部分のテクスチャの2枚をブレンドしています。

赤い部分のテクスチャは常時重ねるのではなく、だいたい0.5秒間隔でブレンドし、燃えているかのようにみせています。

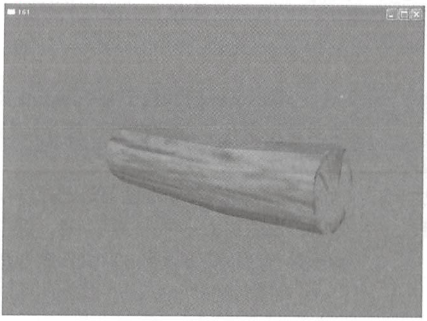
しかし、あまり燃えているようには見えません。単に赤い部分が点滅しているだけで、むしろ非現実的な印象をうけます。

赤い部分は、“表示・非表示”（点滅）ではなく、もっと滑らかに、“強弱”をもたなくてはリアルではありませんが、それはプログラマブルシェーダーを用いれば簡単に実現できます。プログラマブルシェーダーバージョンはTips163にあります。

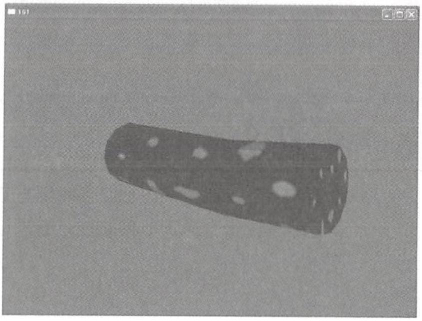
▼実行結果



▼木炭



▼燃えて赤くなっている部分



```
//ステージ0
pDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
pDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
pDevice->SetTextureStageState( 0, D3DTSS_COLOROP,    D3DTOP_MODULATE );
pDevice->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
pDevice->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
// ステージ1
pDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );
pDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );
pDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
pDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
pDevice->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_TEXTURE );
pDevice->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_CURRENT );
//赤い部分のテクスチャをセット
if( (timeGetTime() % 1000) <=500)
{
    pDevice->SetTexture(1,pTexture1);
}
else
{
    pDevice->SetTexture(1,NULL);
}
 for( DWORD i=0; i<dwNumMaterials; i++ )
 {
     pDevice->SetMaterial( &pMeshMaterials[i] );
     pDevice->SetTexture( 0, pMeshTextures[i] );
     pMesh->DrawSubset( i );
 }
```

162　プログラマブルシェーダーにおけるテクスチャの合成

●プログラマブルシェーダーにおけるテクスチャブレンドの方法を理解する　上級

参考サンプルプロジェクト　163　テクスチャで“燃やす”(Pシェーダー版)

仰々しく“テクスチャブレンドの方法”と題しましたが、それほど特殊なものでもなければ難しいことでもありません。

プログラマブルシェーダーではテクスチャでもマテリアルでも、とにかくピクセル”色“を基本単位とするので、統一的にコーディングできます。固定機能の場合は、動作が文字通り固定されているがゆえに意図する動作を行おうとすると逆に前処理が冗長になったり、多くのフラグを覚えなければならないという側面があります。

さて、プログラマブルシェーダーでテクスチャからピクセル（色）を取り出すにはTips160でも述べたとおり、テクスチャ（texsture）とサンプラー（sampler）を定義するわけであり、例えば次のような2つのテクスチャがあるとすると、

```
texture MyTex1;

sampler Sampler1 = sampler_state
{
    Texture = (MyTex1);
    ddressU = Clamp;
    ddressV = Clamp;
};
```

```
texture MyTex2;

sampler Sampler2 = sampler_state
{
    Texture = (MyTex2);
    ddressU = Clamp;
    ddressV = Clamp;
};
```

この2つのブレンドは、いたって簡単です。ピクセルシェーダー内で次のようにすればいいだけのことです。

```
float4 PS( float2 TCoord : TEXCOORD   ) : COLOR
{
    float4 FinalColor= tex2D(Sampler1,TCoord)+tex2D( Sampler2 ,TCoord);
    return FinalColor;
}
```

単に“加算”（場合のよっては乗算することもあるでしょう）するだけです。
tex2D()は、そのサンプラーが示すテクスチャから、そのサンプル方法によってピクセルを取り出します。

＊＊＊＊＊＊

参考までにTips163サンプルのシェーダーコードも次のように、テクスチャをブレンドしています。

```
float4 PS( float2 TCoord : TEXCOORD   ) : COLOR
{
    float4 FinalColor;
    float4 Wood;
    float4 Flame;

    Wood=tex2D(WoodSampler ,TCoord);
    Flame= tex2D(FlameSampler ,TCoord) * Lambency;
    FinalColor = Wood + Flame;

    return FinalColor;
}
```

書き方は僅かに異なりますが、全く同じことを行っていることは明らかでしょう。

One Point

なお、tex1Dは線形テクスチャ、tex3Dはボリュームテクスチャからピクセルを取り出します。

163 プログラマブルシェーダーで「燃やす」

テクスチャの輝度を増減させる

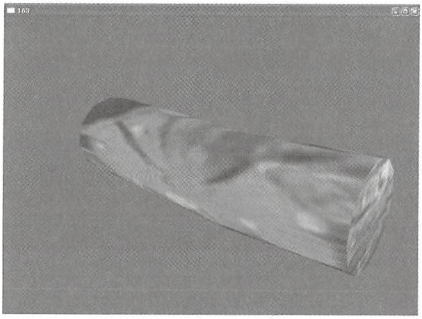
上級

サンプルプロジェクト　163　テクスチャで“燃やす”(Pシェーダー版)

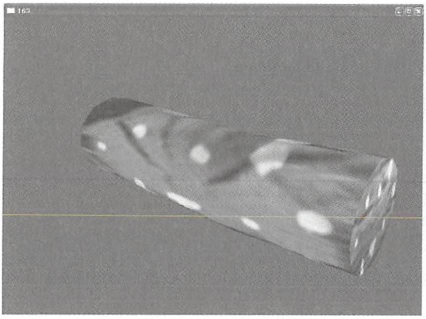
Tips161では、木炭の燃えている部分は、表示されているか、されていないかの2つの状態しか表現できませんでしたが、このサンプルは、滑らかな強弱を表現できています。

ピクセルシェーダーでは、難しいことを行っているわけではなく、時間とともに変化するたった1つの変数を用意し、テクスチャから取り出したピクセルにその変数をかけているだけです。これにより、そのピクセルの輝度を調整できるので、滑らかな強弱となります。

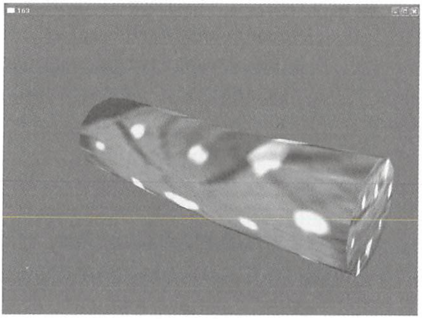
▼実行結果



▼“燃え”の範囲その1



▼“燃え”の範囲その2



●シェーダーファイル側

```
///////////////////////
//ピクセル・シェーダー
///////////////////////
float4 PS( float2 TCoord : TEXCOORD  ) : COLOR
{
    float4 FinalColor;
    float4 Wood;
    float4 Flame;

    Wood=tex2D(WoodSampler ,TCoord);

    Flame= tex2D(FlameSampler ,TCoord) ＊ Lambency;

    FinalColor = Wood + Flame;

    return FinalColor;
}
```

値が変化する変数は、アプリケーション側からシェーダーに与えてやります。

●Main.cpp側

```
static FLOAT Delta=0.02;
 Lambency+=Delta;
 if(Lambency>3 || Lambency<0)
 {
     Delta＊=-1;
 }
 g_pEffect->SetFloat("Lambency",Lambency);
```

One Point

プログラマブルシェーダーは、大抵の場合、固定機能シェーダーよりもシンプルにコーディングできます。

164 プログラマブルシェーダーで「焦がす」

●テクスチャの領域を増減させる

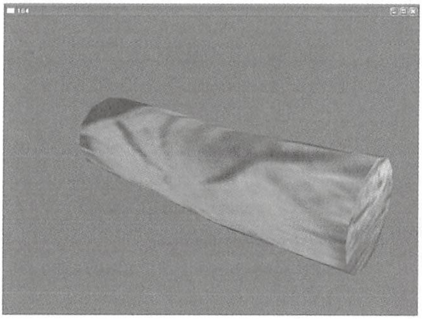
上級

サンプルプロジェクト　164 テクスチャで“焦がす”（シェーダー版）

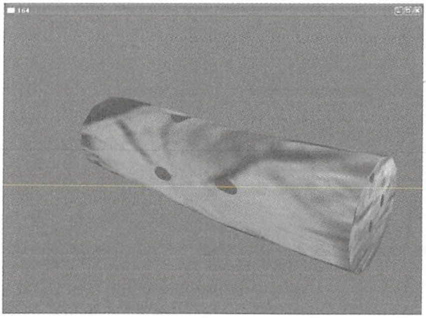
このサンプルでの“焦げ”は、あまりリアルではありませんが、やりたかったことは時間と共に特定の領域が拡大するということです。Tips163では輝度を増減調整させましたが、ここでは2枚目のテクスチャの“範囲”を増減させます。

仕組みはごく単純で、“しきい値”を保持する変数を1個用意し、2枚目のテクスチャのピクセルで、その時点でのしきい値（Threshold）を超えているか、いないかによってその部分のピクセルを表示するかしないかを判断しているだけです。

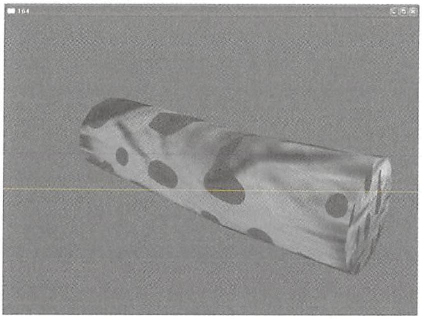
▼実行結果



▼“焦げ”の範囲その1



▼“焦げ”の範囲その2



●シェーダーファイル側

```
////////////////////////
//ピクセル・シェーダー
////////////////////////
float4 PS( float2 TCoord : TEXCOORD  ) : COLOR
{
    float4 FinalColor;
    float4 Wood;
    float4 Burnt;

    Wood= tex2D(WoodSampler ,TCoord);

    Burnt= tex2D(BurntSampler ,TCoord);

    float4 Col=float4(0.2,0.1,0.1,1);//焦げ色

    if(Burnt.r < Threshold)
    {
          FinalColor =  Wood/4+Col*Burnt;
    }
    else
    {
          FinalColor = Wood;
    }

    return FinalColor;
}
```

しきい値は時間とともに変換するので、アプリ側から毎フレームごとに与えてやります。

●Main.cpp側

```
//「焦げの進行具合しきい値」を増減
 static FLOAT Threshold;
 Threshold+=(FLOAT)(Threshold<0.7)*0.001;
 g_pEffect->SetFloat("Threshold",Threshold);
```

One Point

テクスチャ画像側をもうすこし丁寧に作れば（アルファチャンネルの境界をぼかすなど）、もうちょっとましになるはずです。

凹凸感

165 バンプマッピング

●プログラマブルシェーダーによるバンプマッピングを行う

上級

サンプルプロジェクト　165 バンプマッピング

バンプマッピングは、低コストでデコボコ感を表現する優れたテクニックです。低コストというのは、ジオメトリ自体をデコボコにする必要が無いため非常に低ポリゴンでいいからです。実際Tips166では、4ポリゴンの板でも水面のような一見数万ポリゴンもあるかのようなポリゴンに見せています。

バンプマッピングは固定機能でもサポートされていますが、固定機能で実現しようとすると非常にまどろっこしく、コードが妙に複雑になり、本質を学ぶのが難しいと言えます。プログラマブルシェーダーで、どのようにバンプマップを実現するのはを見たほうが、数倍早く理解できます。

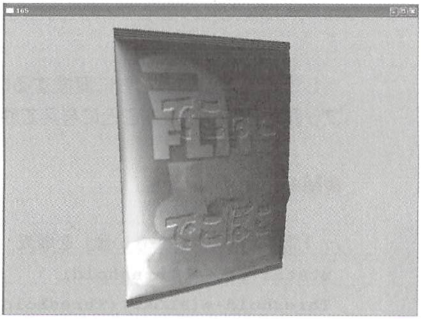
さて、バンプマップはどのような仕組みかというと、ポリゴンごとに座標系を作ります。ライトベクトルをそれぞれの系で変換すれば、どのように反射するかが分かります。出っぱっているポリゴンと平らなポリゴンでは反射が異なるわけですが、その違いはそれら座標系での変換結果で判断できます。

ではそれぞれの座標系はどのように作成するのかというと、ここでは高さマップを基にしています。高さマップ（ハイトマップ）は、通常グレースケール画像で、白と黒の濃淡で凸な部分と凹な部分を表現します。

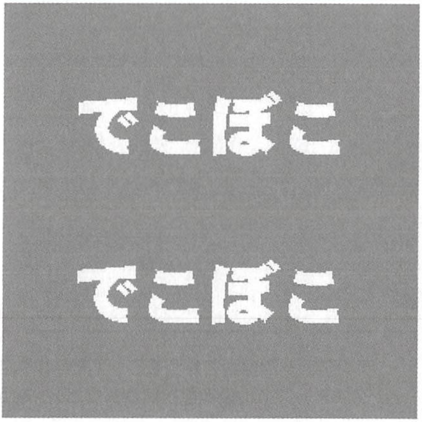
プログラム的には、高さマップをそのまま使用するのではなく、それを基に法線マップを作成しています。

シェーダー側ではその法線マップから従法線（バイ・ノーマル）と接ベクトル（タンジェント・ベクトル）を計算し、ポリゴンごとの系を作成しています。その系の1つ1つ、つまりポリゴンの1つ1つについて反射ベクトルを計算し、最終的なピクセルの濃淡を決定できているというわけです。

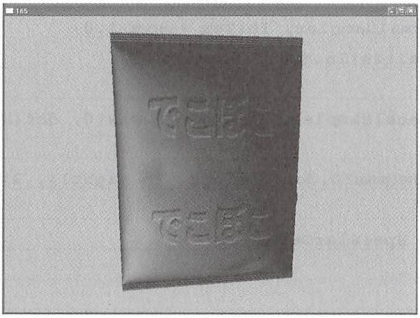
▼実行結果



▼高さマップ



▼デカールテクスチャを隠した状態



●シェーダーファイル

```
///////////////////////
//バーテックス・シェーダー
///////////////////////
VS_OUT VS(float4 Pos : POSITION,float3 Normal : NORMAL,float3 Tangent :
                                     TANGENT0,float2 Texcoord : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos = mul(Pos, matWVP);
    Out.Tex = Texcoord;
    float3 Binormal = cross(Normal,Tangent);

    float3 Eye = vecEyePos - Pos.xyz;
    Out.Eye.x = dot(Eye,Tangent);
    Out.Eye.y = dot(Eye,Binormal);
    Out.Eye.z = dot(Eye,Normal);
    normalize(Out.Eye);

    float3 Light = vecLightDir.xyz;
    Out.Light.x = dot(Light,Tangent);
    Out.Light.y = dot(Light,Binormal);
    Out.Light.z = dot(Light,Normal);
    normalize(Out.Light);

    return Out;
}
///////////////////////
//ピクセル・シェーダー
///////////////////////
float4 PS(VS_OUT In) : COLOR
```

```
{
    float3 Normal = 2.0＊tex2D( NormalSampler, In.Tex ).xyz-1.0;
    float3 Reflect = reflect(-normalize(In.Eye), Normal);

    float4  DiffuseTerm = tex2D( DecalSampler, In.Tex )＊ max(0, dot(Normal,
    In.Light));
    float4 SpecularTerm = 0.7 ＊ pow(max(0,dot(Reflect, In.Light)), 2);

    float4 FinalColor=DiffuseTerm+ SpecularTerm;

    return FinalColor;
}
```

●Entry.cpp側

```
//シェーダー側に各種行列を渡す
 {
     //トランスフォーム行列
     D3DXMATRIX matWVP=matWorld ＊matView＊ matProj;
     g_pEffect->SetMatrix( "matWVP",&matWVP);
    //ライトベクトル
     D3DXMATRIX mat;
     D3DXVECTOR4 vecLight;
     D3DXMatrixInverse( &mat, NULL, &matWorld);
     D3DXVec3Transform( &vecLight, &vecLightPos, &mat );
     D3DXVec3Normalize( (D3DXVECTOR3＊)&vecLight, (D3DXVECTOR3＊)&vecLight
     );
      g_pEffect->SetVector(  "vecLightDir",&vecLight  );

      //視線ベクトル
      D3DXVECTOR4  vecEye;
      mat  =  matWorld  ＊  matView;
      D3DXMatrixInverse(  &mat,  NULL,  &mat);
      D3DXVec3Transform(    &vecEye,&vecEyePt,  &mat  );
      g_pEffect->SetVector(  "vecEyePos",  &vecEye  );
 }
```

ちなみに、本サンプルではメッシュ本来のデカールテクスチャも表示していますが、それをオフにすると、よりバンプがはっきりわかるかもしれません。

次の1行をコメントアウトすればデカールテクスチャが見えなくなります。

```
//g_pEffect->SetTexture("Decal",g_pTex);
```

166 プログラマブルシェーダーで海水面を作る

●プログラマブルシェーダーでのバンプマッピングで表現する

上級

サンプルプロジェクト　166 Pシェーダーで大洋を表現！

本サンプルは、技術的にはバンプマップですので、直前のTipsと同様です。

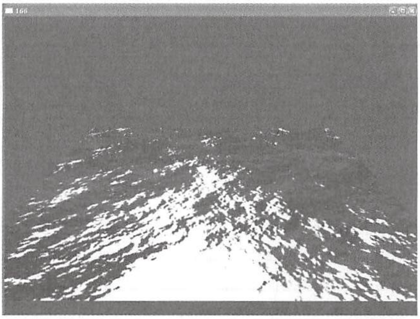
波に動きをつけるため（テクスチャを動かすため）アプリ側からg_fHeightというフロート型の変数を毎フレーム渡しているところが違いといえば僅かな違いです。

バンプマップの利点を生かし、僅か数ポリゴンのジオメトリ上で細かい波を表現しています。図は、16ポリゴンと4ポリゴンのメッシュにバンプマッピングを施した模様です。

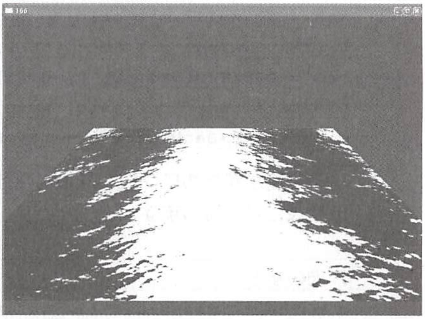
アプリ側のコードのみ解説します。

トランスフォーム行列（ワールド、ビュー、プロジェクションの3つの行列を合成した行列）の逆行列の転置行列をシェーダーに渡します。

▼Water16（実際は32ポリゴン）



▼Water4（実際は8ポリゴン）



```
D3DXMATRIX matWVP=Thing[1].matWorld *matView* matProj;
D3DXMATRIX matWorldInvTranspose;
D3DXMatrixInverse(&matWorldInvTranspose,NULL,&Thing[1].matWorld);
D3DXMatrixTranspose(&matWorldInvTranspose,&matWorldInvTranspose);
```

ビュー行列の逆行列を渡します。

```
D3DXMATRIX matViewInverse;
D3DXMatrixInverse(&matViewInverse,NULL,&matView);
g_pEffect->SetMatrix( "matWorldViewProj",&matWVP);
```

メッシュのワールド行列を渡します。

```
g_pEffect->SetMatrix( "matWorld",&Thing[1].matWorld);
```

ワールド行列の逆行列の転置行列をシェーダーに渡します。

```
g_pEffect->SetMatrixTranspose(  "matWorldInverseTranspose",&matWorldInvTranspose);
g_pEffect->SetMatrix(  "matViewInverse",&matViewInverse);
```

ライトベクトルを渡します。

```
D3DXMATRIX mat;
D3DXVECTOR4 vec;
D3DXVECTOR4 LightPos = D3DXVECTOR4( 0, -10.1, -0.1,0);
D3DXMatrixInverse( &mat, NULL, &Thing[1].matWorld);
D3DXVec4Transform( &vec, &LightPos, &mat );
D3DXVec3Normalize( (D3DXVECTOR3 ＊)&vec, (D3DXVECTOR3 ＊)&vec );
g_pEffect->SetVector( "vecLightDir", &vec );
```

視線ベクトルを渡します。

```
mat = Thing[1].matWorld ＊ matView;
D3DXMatrixInverse( &mat, NULL, &mat);
vec = D3DXVECTOR4( 0, 0, 0, 1);
D3DXVec4Transform( &vec, &vec, &mat );
g_pEffect->SetVector( "vecEyePos", &vec );
```

波を動かすためにグローバルな変数を適当に増減させて渡します。

```
//波の動き
g_pEffect->SetValue("f2Wave",&vecWave,sizeof(vecWave));
vecWave.x+=0.001f;
g_pEffect->SetValue("g_fHeight",&g_fHeight,sizeof(FLOAT));
g_fHeight+=fDelta;
if(g_fHeight<=0.8)
{
      fDelta=0.002;
}
else if(g_fHeight>1.2)
{
     fDelta=-0.002;
}
```

周囲（環境）の映りこみ

167 スフィアマップで擬似的に映りこませる

●球状マッピングにより背景が写り込んでいるように見せる

中級

サンプルプロジェクト　167　スフィアマッピング
サンプルプロジェクト　167　スフィアマッピング（背景つき）
サンプルプロジェクト　167　スフィアマッピングその2
サンプルプロジェクト　167　スフィアマッピングその2（背景つき）

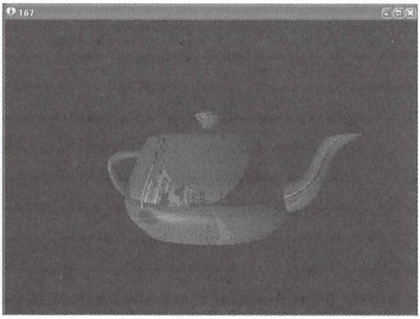
特に3Dソフトのレンダリングを行った人であれば、とりあえず基本的なレンダリングを運用できるようになった次の段階にやりたくなることは、周囲の景色をジオメトリに“写り込ませる”ことではないでしょうか。

リアルタイムCGであるDirect3Dでは当然、3Dソフトのように厳密な写りこみを行わせることはレンダリングコストから考えて現実的ではありません（現在は）。

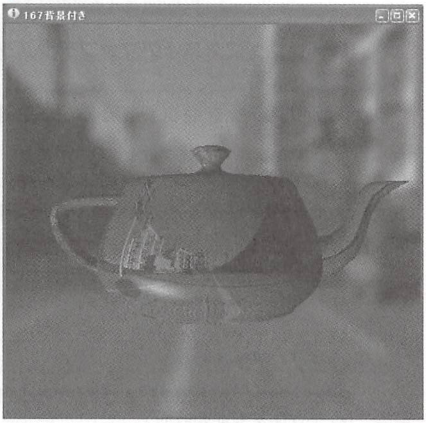
そこで、擬似的に写り込んだように“見せる”トリックの1つがこのスフィアマップです。スフィアマップはTips169で解説するキューブマップよりも高速で実装や仕組みも単純なので、運用しやすいテクニックですが、それゆえ、キューブマップよりも自由が利かずまったくの擬似効果と言えます。

ただ、シーンのスナップショットを見ただけではスフィアマッピングなのかキューブマッピングなのかは判断できません。つまり結果は同じように感じます。

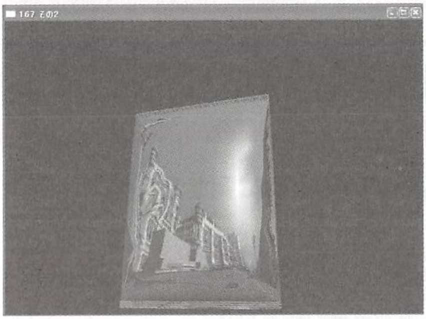
▼実行結果



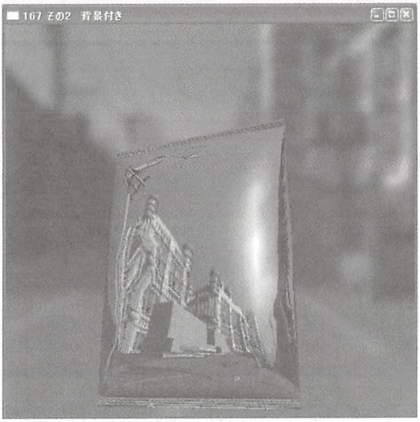
▼実行結果



▼実行結果



▼実行結果



●スフィアマップ用テクスチャのロード

　スフィアマップのテクニックで特殊なのは画像ファイル側であって、テクスチャオブジェクトとしては普通のテクスチャオブジェクトです。

　ロード（テクスチャオブジェクトの作成）は、通常のテクスチャと変わりありません。

```
if( FAILED( D3DXCreateTextureFromFile( pDevice, "SphereMap.bmp",  &pTexture
) ) )
{
    MessageBox(NULL, "テクスチャの読み込みに失敗しました", NULL, MB_OK);
}
```

●メッシュにスフィアマップテクスチャを貼る

　テクスチャをセットするコードも通常どおりです。

　ただし、スフィアマッピングに用のテクスチャ座標を使用するようにパイプラインに指示することがポイントです。

　Direct3D9ではスフィアマップ用のテクスチャ座標を自動生成がサポートされているので楽です。

```
//スフィアマップ用のテクスチャ座標を自動生成するよう指示
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_SPHEREMAP
);
//マテリアルをセット
pDevice->SetMaterial(pTeapotMeshMaterials);
//テクスチャをセット
pDevice->SetTexture(0,pTexture);
pTeapotMesh->DrawSubset(0);
```

実はスフィアマッピングで重要（というか大変）なのは、コーディングではなくテクスチャ画像の作成にあります。それについては次のTips168を参照してください。

One Point

よりリアルな環境マッピングについては、Tips169のキューブマッピングを参照してください。

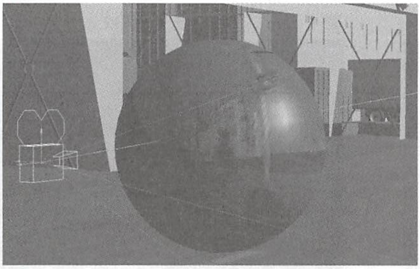
168 スフィアテクスチャを作成する

●スフィアマップになる画像を作成する

中級

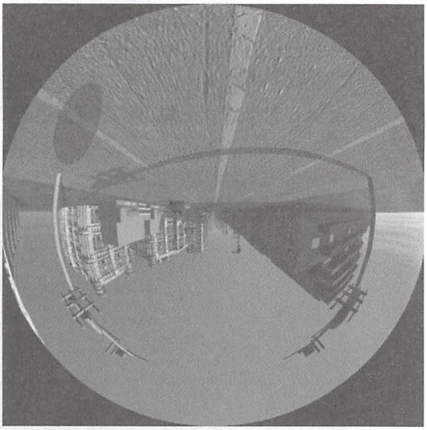
スフィアマップ用のテクスチャ画像は、絵が丸くなっていて周りが黒になっていますが、マッピングされるのは丸の領域だけなので周りの色は関係ありません。

▼実行結果



画像は180度の視野でなければならず、現実に撮影しようとなると個人ではちょっと辛いものがあります。安価に済ませるのであれば球状の鏡に映った風景をカメラで撮影すればいいでしょう。そのような鏡は普通手に入らないので、反射塗料が塗ってある電球でも用は足ります。ただ、その方法だとどうしても鏡に反射したカメラ自体が映ってしまいますが、Photoshopなどの画像編集ソフトでなんとか消すことはできます。

▼テクスチャ画像



面倒なので今回は現実の風景ではなく、3DCGソフトでのレンダリング絵を利用しました。Lightwaveのシーン上に球体を用意します。その球体の反射率を100%にして周囲の風景を写り込ませます。風景を直接カメラに捉えるのではなく、その球に映った風景をカメラに捉えます。3DCGなのでカメラ自体が絵に入ってしまうことはありません。

キューブマッピングでのテクスチャ座標の関係で、絵は上下逆さまにしておきます。

169 キューブマップでリアルに映りこませる

●キューブテクスチャを利用して景色のみならず、他のジオメトリも写りこませる　上級

サンプルプロジェクト　169　キューブマッピング

キューブマッピングは、スフィアマッピングとは異なり単純なテクスチャの貼り付けとは異なります。

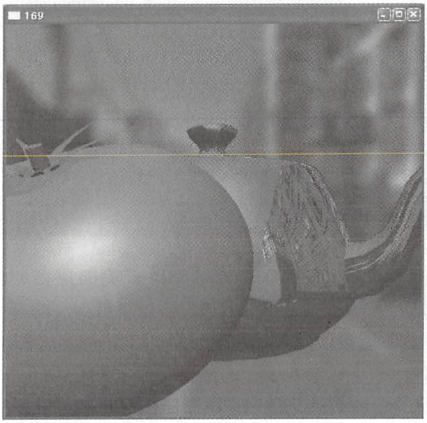
キューブテクスチャと呼ばれる6面（6枚）のテクスチャから成るテクスチャを使用することにより、見る方向により異なる（その方向に見合った）テクスチャが貼られるのを見ることができます。これはどの方向から見てもテクスチャの絵が変化しないスフィアマッピングと大きくことなる点です。

さらに加えて、キューブマッピングでは、“周囲のジオメトリ”までも写りこませることができます。正真正銘の写り込みです。

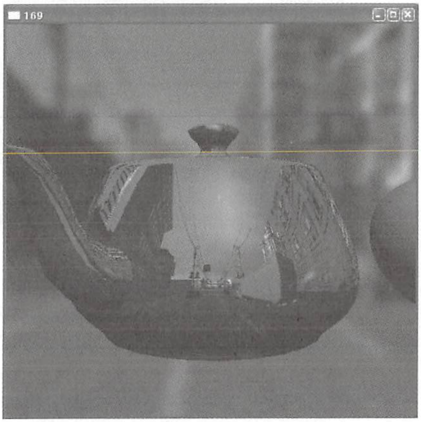
ここでは、キューブマッピング及びキューブテクスチャを利用した他のジオメトリの写り込みを実現するのに超シンプルなコーディングを行っています。マルチパスによる重ね合わせにより、トマトメッシュがうっすらとしか写り込みませんが、コードのシンプルさを優先しました。

なにかこう心霊写真（？）のようにうっすらとしか写っていないので、分かりづらいかもしれません。分かりやすいように背景用のテクスチャをオフにすると、トマトがはっきり写り込んでいるのが分かります。

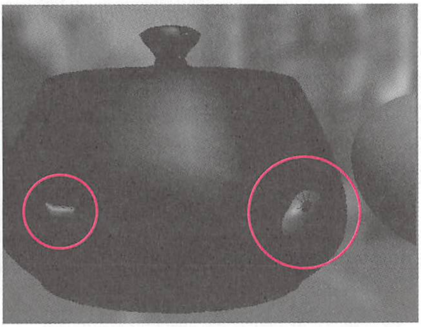
▼キューブテクスチャその1



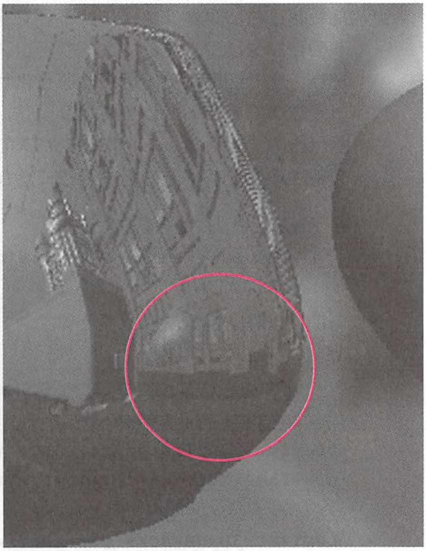
▼キューブテクスチャその2



▼背景用のテクスチャをオフ



▼写り込んだトマトメッシュ



コードの要点部分はポイントは次のとおりです。

```
//" 背景のキューブマッピング用" キューブマップイメージを読み込む
if(FAILED(D3DXCreateCubeTextureFromFileEx(pDevice,"Town.dds",D3DX_DEFAULT,D3
DX_DEFAULT,
    D3DUSAGE_RENDERTARGET,ds.Format,D3DPOOL_DEFAULT,D3DX_DEFAULT,
    D3DX_DEFAULT,0xFF000000,NULL,NULL,&pCubeEnvMap)))
{
    MessageBox(0,"キューブテクスチャ作成失敗","",MB_OK);
    return  E_FAIL;
}
```

背景のキューブマップ用の画像から、背景用のキューブテクスチャを作成します。

```
//" 周囲のメッシュのレンダリング先" としてのキューブマップイメージを読み込む
if(FAILED(pDevice->CreateCubeTexture(CUBEMAP_DIMENSION,0,D3DUSAGE_RENDER
TARGET,ds.Format,
    D3DPOOL_DEFAULT,&pCubeRender,NULL)))
{
    MessageBox(0,"キューブテクスチャ作成失敗","",MB_OK);
    return  E_FAIL;
}
```

もう1枚キューブテクスチャを作成します。ただしこれに画像ファイルは不要です。トマトのレンダリング先のキューブテクスチャであり、絵で初期化する必要はありません。真っ黒な中にトマトをレンダリングし、それを絵付きのティーポットとマルチパスで重ねるの絵は不用です。そもそもそこに絵があっても、毎回そのキューブテクスチャのサーフェイスをクリアするので（クリアしないとトマトの残像が累積します）どっちみちすぐに黒くなりわけで、意味がありません。

メッシュの法線を初期化するように指示します。

```
//メッシュの法線を自動的に正規化
        pDevice->SetRenderState(D3DRS_NORMALIZENORMALS,TRUE);
```

これ以降はレンダリングです。

```
// 現在のビュー行列とプロジェクション行列を保存しておく（この後一時的に変更するため）
D3DXMATRIX matProjHold, matViewHold;
pDevice->GetTransform(D3DTS_VIEW, &matViewHold);
pDevice->GetTransform(D3DTS_PROJECTION, &matProjHold);
// バックバッファを得る
LPDIRECT3DSURFACE9 pBackBuffer;
pDevice->GetRenderTarget(0,&pBackBuffer);
// キューブマップ上へのレンダリング用にプロジェクション行列を作成
D3DXMATRIX matProj;
D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI/2, 1.0f, 0.01f, 1000.0f);
pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
// キューブ マップ内、 6 つの面それぞれに見合ったビュー行列を作成
for(DWORD i=0; i<6; i++)
{
    D3DXVECTOR3 vecEye = D3DXVECTOR3(0.0f, 0.0f, 0.0f);
    D3DXVECTOR3 vecLookat, vecUp;
    switch(i)
    {
    case D3DCUBEMAP_FACE_POSITIVE_X:
        vecLookat = D3DXVECTOR3(1.0f, 0.0f, 0.0f);
        vecUp = D3DXVECTOR3(0.0f, 1.0f, 0.0f);
        break;
    case D3DCUBEMAP_FACE_NEGATIVE_X:
        vecLookat = D3DXVECTOR3(-1.0f, 0.0f, 0.0f);
        vecUp = D3DXVECTOR3( 0.0f, 1.0f, 0.0f);
        break;
    case D3DCUBEMAP_FACE_POSITIVE_Y:
        vecLookat = D3DXVECTOR3(0.0f, 1.0f, 0.0f);
        vecUp = D3DXVECTOR3(0.0f, 0.0f,-1.0f);
        break;
    case D3DCUBEMAP_FACE_NEGATIVE_Y:
        vecLookat = D3DXVECTOR3(0.0f,-1.0f, 0.0f);
        vecUp = D3DXVECTOR3(0.0f, 0.0f, 1.0f);
        break;
    case D3DCUBEMAP_FACE_POSITIVE_Z:
        vecLookat = D3DXVECTOR3( 0.0f, 0.0f, 1.0f);
        vecUp = D3DXVECTOR3( 0.0f, 1.0f, 0.0f);
        break;
    case D3DCUBEMAP_FACE_NEGATIVE_Z:
        vecLookat = D3DXVECTOR3(0.0f, 0.0f,-1.0f);
        vecUp = D3DXVECTOR3(0.0f, 1.0f, 0.0f);
        break;
    }
    D3DXMATRIX matView;
    D3DXMatrixLookAtLH(&matView, &vecEye, &vecLookat, &vecUp);
    pDevice->SetTransform(D3DTS_VIEW, &matView);
    // キューブマップ内の任意の面のサーフェイスを得る（それをレンダリングターゲットとして設
    定する）
    LPDIRECT3DSURFACE9 pCubeFace;
    pCubeRender->GetCubeMapSurface((D3DCUBEMAP_FACES)i, 0, &pCubeFace);
    pDevice->SetRenderTarget (0,pCubeFace );
    SAFE_RELEASE(pCubeFace);
    //写り込むトマトメッシュを”キューブテクスチャ上に”レンダリング
    {
        pDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );
        pDevice->SetTransform( D3DTS_WORLD, &matWorldTomato );
        for( DWORD i=0; i<dwNumMaterials; i++ )
        {
            pDevice->SetMaterial( &pMeshMaterials[i] );
            pDevice->SetTexture( 0, pMeshTextures[i] );
            pMesh->DrawSubset( i );
        }
    }
}
// レンダリングターゲットを元に戻す
pDevice->SetRenderTarget(0,pBackBuffer);
SAFE_RELEASE(pBackBuffer);
// 保存しておいたビュー行列とプロジェクション行列により、変換行列を元に戻す
pDevice->SetTransform(D3DTS_VIEW, &matViewHold);
pDevice->SetTransform(D3DTS_PROJECTION, &matProjHold);
```

```
//キューブマップ用のテクスチャ座標を自動生成するよう指示
pDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX,D3DTSS_TCI_CAMERA
SPACEREFLECTIONVECTOR );
```

この後で、ティーポットメッシュをレンダリングします。それからトマトメッシュのレンダリングをもう一度行います。ただし今度はテクスチャにではなく通常のレンダリングターゲット上に普通にレンダリングします。このレンダリングは外周を回っているトマトになります。

170 キューブテクスチャ用の画像を作成する

●キューブテクスチャの絵となる画像の作成方法

中級

キューブテクスチャ用の画像（.ddsファイル）は特殊なもので、普通の画像編集ソフトで作成できません。DirectX　SDKには、DxTex.exeという（フリーの）テクスチャ作成用のツールが同梱されています。キューブテクスチャもDxTexで作成します。

サイコロ状のような正6面体の各面全てについて絵を用意します。したがって6枚の絵が必要になります。キューブマップ用画像の作成の時と同様、キューブマップ画像もなかなか実際に撮影するとなると苦労します。

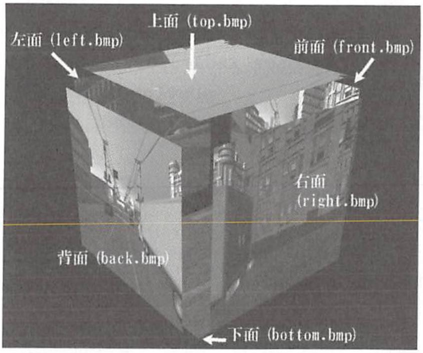
ここではキューブマップの時と同じく、Lightwaveによるレンダリング絵を利用します。ある視点から上下、左右、前後の6方向についての景色を作ります。ファイル名をそのまま方向にすると分かりやすいでしょう。

例えば次のように各方向の景色をそれぞれ同じサイズのビットマップで作成したとしましょう。

top.bmp	上面用
bottom.bmp	下面用
left.bmp	左面用
right.bmp	右面用
front.bmp	前面用
back.bmp	背面用

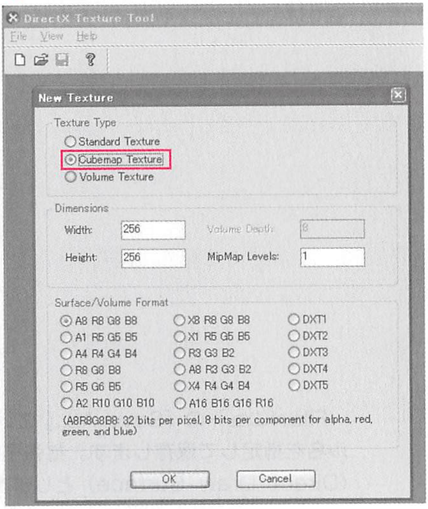
これら6枚のビットマップをDxTex.exeで1つのキューブテクスチャに合体します。

▼実行結果



まず最初に行う操作は、新規のキューブテクスチャ（まだ空っぽの状態です）を作成します。Fileメニューの「New　Texture」をクリックすると次のダイアログボックスが出ますので、「Texture　Type」では「Cubumap　Texture」のラジオボタンを選択してください。

▼Cubumap　Texture



New Texture

Texture Type
○ Standard Texture
◉ Cubemap Texture
○ Volume Texture

Dimensions
Width: 256　Volume Depth: 8
Height: 256　MipMap Levels: 1

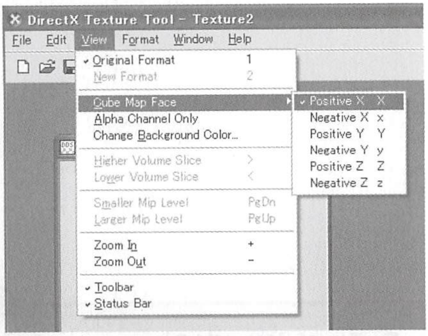
Surface/Volume Format
◉ A8 R8 G8 B8　○ X8 R8 G8 B8　○ DXT1
○ A1 R5 G5 B5　○ X1 R5 G5 B5　○ DXT2
○ A4 R4 G4 B4　○ R3 G3 B2　○ DXT3
○ R8 G8 B8　○ A8 R3 G3 B2　○ DXT4
○ R5 G6 B5　○ X4 R4 G4 B4　○ DXT5
○ A2 R10 G10 B10　○ A16 B16 G16 R16
(A8R8G8B8: 32 bits per pixel, 8 bits per component for alpha, red, green, and blue)

DirectX Texture Tool
File
View
Help
OK
Cancel

新規作成したら、次に“6面のうちどの面を読み込むか”を指定します。

Viewメニューの「Cube Map Face」をクリックしてから特定の面を指定します。右の図ではPositiveX（X軸正方向の面）を指定しています。

▼Cube　Map　Face

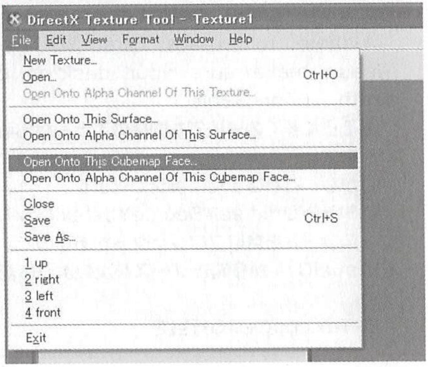


DirectX Texture Tool - Texture2
File
Edit
View
Format
Window
Help
✓ Original Format　1
New Format　2
Cube Map Face
Alpha Channel Only
Change Background Color...
Higher Volume Slice　>
Lower Volume Slice　<
Smaller Mip Level　PgDn
Larger Mip Level　PgUp
Zoom In　+
Zoom Out　-
✓ Toolbar
✓ Status Bar
✓ Positive X　X
Negative X　x
Positive Y　Y
Negative Y　y
Positive Z　Z
Negative Z　z

X軸正方向の面（要するに右面）の絵として対応した画像を読み込みます。

Fileメニューの「Open Onto Cubemap Face」をクリックします。この面用の画像はright.bmpです。

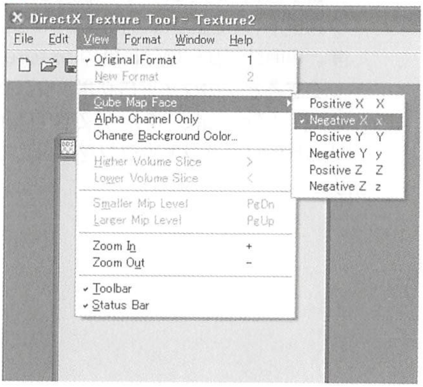
▼Open　Onto　Cubemap　Face



DirectX Texture Tool - Texture1
File
Edit
View
Format
Window
Help
New Texture...
Open...　Ctrl+O
Open Onto Alpha Channel Of This Texture...
Open Onto This Surface...
Open Onto Alpha Channel Of This Surface...
Open Onto This Cubemap Face...
Open Onto Alpha Channel Of This Cubemap Face...
Close
Save　Ctrl+S
Save As...
1 up
2 right
3 left
4 front
Exit

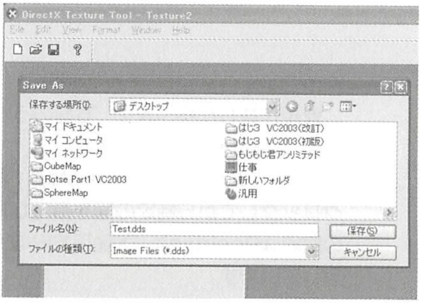
以降、残り5面について同様の操作を繰り返せばでき上がりです。

▼Open Onto Cubemap Face


DirectX Texture Tool - Texture2
File
Edit
View
Format
Window
Help
Original Format 1
New Format 2
Cube Map Face
Alpha Channel Only
Change Background Color...
Higher Volume Slice >
Lower Volume Slice <
Smaller Mip Level PgDn
Larger Mip Level PgUp
Zoom In +
Zoom Out -
Toolbar
Status Bar
Positive X X
Negative X x
Positive Y Y
Negative Y y
Positive Z Z
Negative Z z

Fileメニューの「Save As..」で、ファイル名を指定して保存します。拡張子は.dds（Direct Draw Surface）とします。

▼ファイル名を指定して保存


DirectX Texture Tool - Texture2
File
Edit
View
Format
Window
Help
Save As
保存する場所(I):
デスクトップ
マイ ドキュメント
マイ コンピュータ
マイ ネットワーク
CubeMap
Rotse Part1 VC2003
SphereMap
はじ3 VC2003(改訂)
はじ3 VC2003(初版)
もじもじ君アンリミテッド
仕事
新しいフォルダ
汎用
保存(S)
キャンセル

ファイル名(N): Test.dds

ファイルの種類(T): Image Files (*.dds)

DxTexが起動しない？

最近、筆者の環境においてDxTexが突然次のようなメッセージを出して起動しなくなりました。

```
Unable to create Direct3D Device
Please make sure your desktop color depth 16 or 32bit...
```

おそらく多くの環境でも同様のエラーが出ると思われます。

しかし、これはすぐに解決できます。

C:SDK¥Utilities¥Source¥dxtexにDxTexのソースコードを含むプロジェクトがあるので、それをVisualC++から開きソースを次のように変更します。

DxTex.cppの215行目を

```
devType = D3DDEVTYPE_REF;
```

から

```
devType = D3DDEVTYPE_HAL;
```

にしてビルドしてください。その後はビルドされてでき上がったDxTexの実行ファイルを使うようにすればいいでしょう。

なお、この不具合はAugast2005以前のSDKに付属のDexTexで発生するようです。December2005ではこの問題が静かに解決（！）されていました。したがって本書付属のSDKであれば大丈夫であり、もし読者のDxTexが同様の不具合を起こしている場合、そのDxTexは古いかもしれません。

171 キューブテクスチャを作成・初期化する

●IDirect3DCubeTexture9オブジェクトを作成する

初級

参考サンプルプロジェクト　169 キューブマッピング

引数にはそれぞれにおいてさまざまなタイプがあるので、一般的な例を書きます。

```
#define CUBEMAP_DIMENSION 512
```

必須ではないですが、視認性のためにキューブテクスチャのサイズを一応定数定義しておきます。これは512ピクセルという意味です。キューブテクスチャは6面すべてが同じサイズでなければなりません。それはすべてが正方形でなければならないということでもあります。（長方形で各面が同じサイズの6面体は作れませんよね）

したがって、サイズは一種類だけでいいわけです。

```
LPDIRECT3DCUBETEXTURE9 pCubeEnvMap=NULL;
```

キューブテクスチャオブジェクトのポインターを用意します。

●ddsファイルからキューブテクスチャオブジェクトを作成（Town.ddsという画像ファイルを読み込む場合）

通常の用途であれば、ddsファイルから作成するのでほとんどはこの関数を利用することかと思います。

```
D3DDISPLAYMODE  ds;
pDevice->GetDisplayMode(0,&ds);

D3DXCreateCubeTextureFromFileEx(pDevice,"Town.dds",D3DX_DEFAULT,D3DX_DEFAULT
,D3DUSAGE_RENDERTARGET,ds.Format,D3DPOOL_DEFAULT,D3DX_DEFAULT, D3DX_DEFAULT,
0xFF000000,NULL,NULL,&pCubeEnvMap);
```

●空（から）のキューブテクスチャオブジェクトの作成

空のオブジェクト（絵が不用）でもいい場合があります。Tips169がいい例です。ジオメトリのレンダリング先としてキューブテクスチャを利用する場合、絵が書いてあっても結局クリアすることになるので意味がないからです。

```
D3DDISPLAYMODE  ds;
pDevice->GetDisplayMode(0,&ds);

pDevice->CreateCubeTexture(CUBEMAP_DIMENSION,0,D3DUSAGE_RENDERTARGET,
ds.Format,D3DPOOL_DEFAULT,&pCubeRender,NULL);
```

プログラマブルシェーダー共通基本処理

172 最小機能で動くシェーダーを書く

●とにかく超シンプルなプログラマブルシェーダーで流れを理解する　上級

サンプルプロジェクト　172　最小Pシェーダー

本当に“最小”にしようと思えば、頂点シェーダーかピクセルシェーダーのどちらかのみ書くことになりますが、それでは仕組みを掴めず、見やすさのために最小コードにするという目的をかえって失ってしまいます。

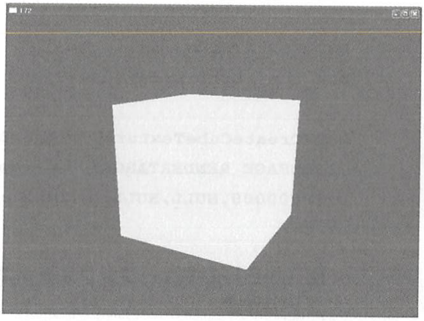
ここで解説するのは、とりあえず頂点シェーダーからピクセルシェーダーへのデータの受け渡しを行う、ちゃんとしたもので最小ということです。

頂点シェーダーは、パイプラインから主にジオメトリに関る情報を受け取り、3次元的な処理を施すフェーズです。ジオメトリとは、頂点のことです。頂点の座標、色、法線、接線などの情報を受け取ることができ、コーディング次第でそれらを自由に操れます。

頂点シェーダーで3次元部分の処理が終われば、次に最終的な2次元ピクセルのフェーズに入ります、それがピクセルシェーダーの仕事です。したがって頂点シェーダーでの処理がまずいとピクセルシェーダーにも影響が及び、最終的に妙な絵になることがあります。

ここでは、メッシュの3D変換を行い、その後レンダリングするという単純なことをピクセルシェーダーで行います。この程度なら固定機能シェーダーでやってもいいですが、シンプルな処理というとこれくらいしかありません。

▼実行結果



●Entry.cpp側のコード

```
LPD3DXEFFECT g_pEffect=NULL;

//シェーダーを読み込む
if( FAILED(  D3DXCreateEffectFromFile(pDevice, "最小.fx", NULL, NULL,
    0, NULL, &g_pEffect, NULL)))
{
    MessageBox( NULL, "シェーダーファイル読み込み失敗", "", MB_OK);
    return E_FAIL;
}

//レンダリング
 if( SUCCEEDED( pDevice->BeginScene() ) )
 {
     g_pEffect->SetTechnique( "tecMinimum" );
     D3DXMATRIX matAll = matWorld * matView * matProj;
     g_pEffect->SetMatrix( "matWVP", &matAll );
     g_pEffect->Begin( NULL, 0 );
     g_pEffect->BeginPass(0);
     for( DWORD i=0; i<dwNumMaterials; i++ )
     {
            g_pEffect->SetTexture("texMinimum",pMeshTextures[i]);
            pMesh->DrawSubset( i );

     }
    g_pEffect->EndPass();
    g_pEffect->End();
    pDevice->EndScene();
 }
```

●シェーダー側のコード

コメントと中括弧のせいで行数が多く見えますが、シェーダーの実質的行数はわずか5行です。

アプリケーション側（Entry.cpp）から何らかの値をシェーダーに伝えるには、シェーダー側で変数をグローバルに宣言します。

ここでは、ワールド変換から最後の射影変換までの変換を行う行列を宣言しています。

```
float4x4 matWVP; //ワールドから→ビュー→プロジェクションまでの3D変換行列
```

3D変換、すなわちワールド、ビュー、プロジェクションの3つの変換を行います。

```
//バーテックス・シェーダー
float4 VertexShader(float4 f4Position : POSITION) : POSITION
{
    return mul( f4Position, matWVP );//3D変換
}
```

頂点シェーダーで処理された出力結果はピクセルシェーダーの入力になります。そして、ピクセルシェーダーの戻り値は、最終的にモニターに表示されるピクセルの色です。

```
//ピクセル・シェーダー
float4 PixelShader() : COLOR
{
    return float4(1,1,0,1); //黄色単色
}
```

テクニックにはパス（1回のレンダリング）を記述します。パスの中では、頂点シェーダー、ピクセルシェーダーをどのようにコンパイルし、どの関数（ここではVertexShader関数とPixelShader関数しかないですが）を使用するかを指定します。関数は複数あっても構いません。

またパス自体も複数あっても構いません。もっと言えば1つのシェーダーファイル内にテクニック自体を複数書いても構いません。

```
// テクニック
technique tecMinimum
{
    pass P0
    {
        VertexShader = compile vs_1_1 VertexShader();
        PixelShader  = compile ps_1_1 PixelShader();
    }
}
```

One Point

ピクセルシェーダーの「return　float4(1,1,0,1);　//黄色単色」のRGBの割合をいろいろ変えて結果の違いを見てみてください。いろいろいじってみることが理解の近道です。

173 ディフューズ色くらいは反映したい

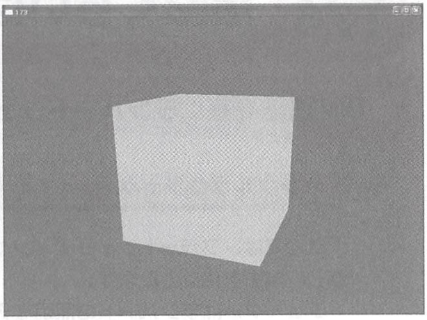
メッシュのディフューズ成分をシェーダーに渡す

上級

サンプルプロジェクト　173　ディフューズの反映

ほとんど直前のTipsと同様です。直前のサンプルでは、メッシュがどのようなマテリアルであろうが、問答無用で黄色一色のフラットシェーディングになってしまいましたが、ここではメッシュがもつマテリアル（のディフューズ色）でフラットシェードするので“多少”ましになっています。

▼実行結果



●Entry.cpp側のコード

```
for( DWORD i=0; i<dwNumMaterials; i++ )
 {
        //メッシュのディフューズ色を渡す
        {
                g_pEffect->SetValue("Diffuse",(LPVOID)&pMeshMaterials[i].
                Diffuse,16);
        }
        g_pEffect->SetTexture("texMinimum",pMeshTextures[i]);
        pMesh->DrawSubset( i );
 }
```

シェーダーに値を渡すためのメソッドはいくつかありますが、ここではSetValueメソッドを使います。

第1引数はシェーダー内のグローバル変数名、第2引数はデータのポインタ、第3引数はバイト数です。ディフューズはFLOAT値が4つなので、4×4で16バイトになります。

●シェーダー側のコード

```
float4 Diffuse;

float4 PixelShader() : COLOR
{
    return Diffuse; //渡されたメッシュのディフューズ色を出力
}
```

単純に、受け取ったディフューズを出力します。

174 陰影くらいは付けたい

●頂点法線を基にランバートシェーディングする

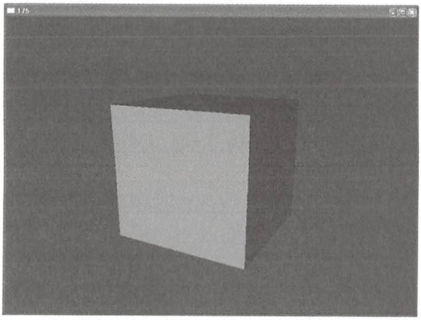
上級

サンプルプロジェクト　174　陰影くらいは付けたい

ここでは、フラットシェーディングから脱却し、陰影を付けてみます。

シェーダーはランバートの照明モデルに沿ってコーディングしていますが、環境光の項は省いています。

▼実行結果



●シェーダー側のコード

```
////////////////////////
//型定義
////////////////////////
struct  VS_OUT
{
        float4  Position        :  POSITION;
        float4  Color           :  COLOR0;
};

////////////////////////
//グローバル
////////////////////////
```

```
float4x4  WVP;  //ワールドから射影までの変換行列
float4x4  WIT;    //ワールド行列の逆行列の転置行列
float4  LightDir;    //「ライトの方向ベクトル
float4  LightIntensity;  //入射光の強度
float4  Diffuse;  //拡散反射率

//////////////////////
//バーテックス・シェーダー
//////////////////////
VS_OUT  VS(float4  Position  :  POSITION,float3  Normal  :  NORMAL)
{
    VS_OUT  Out  =  (VS_OUT)0;

    Out.Position=mul(Position,WVP);

    float3  WNormal=normalize(mul(Normal,(float3x3)WIT));

    Out.Color=LightIntensity  *  Diffuse  *  dot(WNormal,LightDir);//この式はランバートの余弦則だけですが、通常、ランバート照明モデルと言えば環境光の項を含めることが多いようです

    return  Out;
}
//////////////////////
//ピクセル・シェーダー
//////////////////////
float4  PS(VS_OUT  In)  :  COLOR
{
    return  In.Color;
}

//////////////////////
//  テクニック
//////////////////////
technique  tecLambert
{
        pass  Lambert
        {
                VertexShader  =  compile  vs_2_0  VS();
                PixelShader   =  compile  ps_2_0  PS();
        }
}
```

175 シェーダーにデータを渡す

●ID3DXEffect::SetXXX系のメソッドを使う　上級

参考サンプルプロジェクト　172　173　174　の各サンプル

シェーダーになんらかのデータを渡すために用意されているメソッドはけっこうあります。

●変数および変数の配列を渡す場合は、

SetBoolメソッド　ブール値
SetBoolArrayメソッド　ブール値の配列
SetFloatメソッド　浮動小数点数値
SetFloatArrayメソッド　浮動小数点数値の配列
SetIntメソッド　整数
SetIntArrayメソッド　整数の配列

●ベクトルおよびベクトルの配列を渡す場合は、

SetVectorメソッド
SetVectorArrayメソッド　ベクトルの配列

●行列を渡すには、

SetMatrixメソッド　普通の行列
SetMatrixArrayメソッド　普通の行列の配列
SetMatrixPointerArrayメソッド　行列のポインタの配列
SetMatrixTransposeメソッド　転置行列
SetMatrixTransposeArrayメソッド　転置行列の配列
SetMatrixTransposePointerArrayメソッド　転置行列のポインタの配列

●テクスチャを渡すには、

SetTextureメソッド

しかし、ほとんどはSetValueというメソッドで代用できるので全て憶える必要はありません。

SetValueの第1引数はシェーダー内のグローバル変数名、第2引数はデータのポインタ、第3引数はバイト数です。

例えばINT型データを渡すのであれば、

```
INT i;
g_pEffect->SetValue("（シェーダー内での変数名）",(LPVOID)&i ,4);  INT型は4バイト
```

BOOL型データを渡すのであれば、

```
bool b;
g_pEffect->SetValue("（シェーダー内での変数名）",(LPVOID)&b ,1); bool型（BOOL型で
はない）は1バイト
```

INT型の配列データを渡すのであれば、

```
INT i[10];
g_pEffect->SetValue("（シェーダー内での変数名）",(LPVOID)i ,40); （要素10個×4バイ
ト）
```

などとすればいいわけです。わざわざ型や配列ごとにメソッドがありますが、不要とも思えます。

ただし、テクスチャを渡す場合はSetTextureを使用しなくてはなりません。

176 シェーダーからデータを取得する

●ID3DXEffect::GetXXX系のメソッドを使用して、シェーダーから何らかのデータを受け取る 上級

直前のTipsと逆の処理です。直前のTipsではSetXXXというメソッドでデータを渡すということを述べました。データを取得するには、そのSetをGetに変えればいいだけです。例えば、SetIntならGetInt、SetValueならGetValueという具合です。

SetXXX系のメソッドが理解できればGetXXX系のメソッドも理解したも同然です。Get系のメソッドでも、特にGetValueメソッドは汎用性のあるメソッドで、他のほとんど全てのメソッドの代わりに使えます。

177 テクスチャとサンプリングを設定する

シェーダーではテクスチャとサンプラーが対で必要

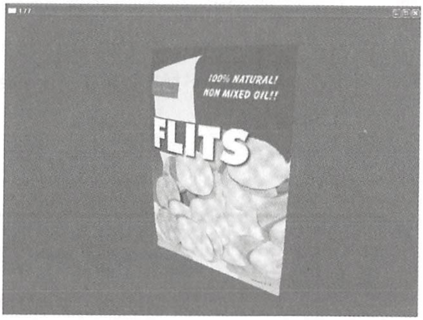
上級

サンプルプロジェクト 177 テクスチャの扱い方

ここでは、シェーダーにおけるテクスチャ関連のコーディングを解説します。

固定機能よりも、ピクセルを操作する様が見えるので、テクスチャがどのように最終ピクセル（最終色）に関るのかという原理を理解する良い機会だと思います。

▼実行結果



シェーダーファイル側

```
texture texDecal; //メッシュのテクスチャ

sampler Sampler = sampler_state //サンプラー
{
    Texture = <texDecal>;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = NONE;
    AddressU = Clamp;
    AddressV = Clamp;
};

struct VS_OUT
{
    float4 f4Position : POSITION;
    float2 f2Texture : TEXCOORD0;
};

////////////////////////
//バーテックス・シェーダー
////////////////////////
```

```
VS_OUT VertexShader(float4 f4Position : POSITION , float2 f2Texture :
TEXCOORD)
{
    VS_OUT Out;
    Out.f4Position = mul( f4Position, matWVP );
    Out.f2Texture = f2Texture;
    return Out;
}

///////////////////////
//ピクセル・シェーダー
///////////////////////
float4 PixelShader( float2 f2Texture : TEXCOORD ) : COLOR
{
    return tex2D( Sampler, f2Texture );
}
```

シェーダーではテクスチャ（texture）とサンプラー（sampler）が対で必要です。

基本的な考え方は、どのような絵（テクスチャ）をどのように貼る（サンプラー）という考え方です。サンプラーはテクスチャの貼り方を表しています。

ここでは、一般的なデカールテクスチャのサンプルタイプを書きましたが、もちろん様々なサンプルが可能です。

●Entry.cpp側

シェーダーインターフェイス内の`ID3DXEffect::SetTexture`メソッドによりセットします。

```
for( DWORD i=0; i<dwNumMaterials; i++ )
 {
    g_pEffect->SetTexture("texDecal",pMeshTextures[i]);
    pMesh->DrawSubset( i );
 }
```

アルファブレンディング

178 アルファブレンディングする

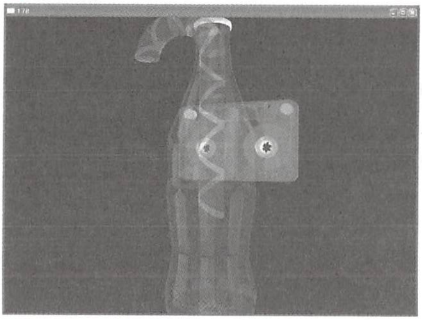
物体を透明、半透明でレンダリングする

中級

サンプルプロジェクト　178 アルファブレンディング

アルファブレンディングを行うには、次のコード片の太字部分のように書きます。

▼実行結果



```
for( DWORD i=0; i<pThing->dwNumMaterials; i++ )
{
      //アルファブレンディングを設定する
      pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
      pDevice->SetRenderState ( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
      pDevice->SetRenderState ( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
      pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1,D3DTA_DIFFUSE);
      pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG2,D3DTA_DIFFUSE);

      pDevice->SetMaterial( &pThing->pMeshMaterials[i] );
      pDevice->SetTexture( 0,pThing->pMeshTextures[i] );

      pThing->pMesh->DrawSubset( i );
}
```

SetRenderStateはフレームバッファアルファの設定です。

そしてSetTextureStageStateは、テクスチャアルファの設定です。このサンプルでは「キャンディメッシュ」がテクスチャを使っているので、テクスチャアルファの設定もしないと透けてくれません。

179 カメラ位置に依存しないアルファブレンディングを実現する

●アルファブレンドされない視点を解消する

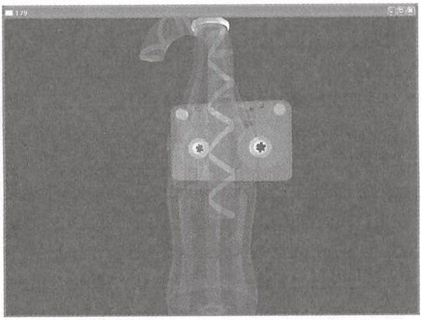
上級

サンプルプロジェクト　179 カメラ位置に依存しない

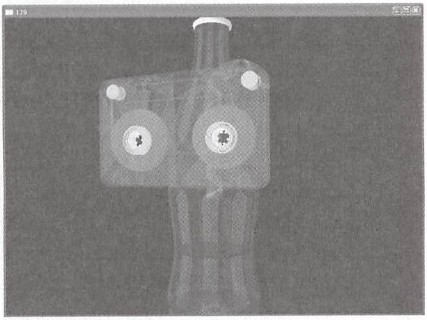
アルファブレンドは、ある特定の視点からはきちんとブレンドされていてもカメラを反対側にもっていった場合、なにも工夫をしないとうまくブレンドされません。これは、アルファブレンディングの仕組みそのものからくる不具合で、仕方のないことです。

この不具合を解消するには、その都度メッシュを“カメラ座標系での位置（奥行き）”でソートして、順番を変えながらレンダリングします。

▼実行前



▼実行結果



qsort関数でソートし、並べ替えた順番でレンダリングします。

```
if( SUCCEEDED( pDevice->BeginScene() ) )
 {
    THING＊ pThing[THING_AMOUNT+1];

    for(DWORD i=0;i<THING_AMOUNT;i++)
    {
           pThing[i]=&Thing[i];D3DXVec3TransformCoord(&pThing[i]->vecLook,
           &pThing[i]->vecPosition,&pThing[i]->matWorld);
           D3DXVec3TransformCoord(&pThing[i]->vecLook,&pThing[i]->vec
           Position,&matView);
    }
    qsort(pThing,THING_AMOUNT,sizeof(THING＊),Compare);
```

```
        for(DWORD i=0;i<THING_AMOUNT;i++)
        {
                RenderThing(pThing[i] );
        }
         pDevice->EndScene();
    }
```

qsort関数で使うコールバック関数を実装します。

```
INT Compare( const VOID＊ a, const VOID＊ b )
{
        THING＊＊ ppThingA=(THING＊＊)a;
        THING＊＊ ppThingB=(THING＊＊)b;

        if((＊ppThingA)->vecLook.z<(＊ppThingB)->vecLook.z)
        {
                return 1;
        }
        else if((＊ppThingA)->vecLook.z>(＊ppThingB)->vecLook.z)
        {
                return -1;
        }
        else
        {
                return 0;
        }
}
```

One Point

ソートのキーにするのは“カメラ座標系での奥行き（Z成分）”です。言い換えると、あくまでその時見えている状態での奥行きです。ワールド座標系での位置ではないことに注意してください。

180 マテリアルのアルファ成分を反映させる

マテリアルのアルファ成分で透明度を調整する　中級

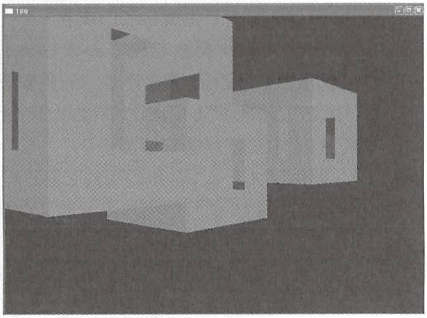
サンプルプロジェクト　180　マテリアルを利用

　1つのメッシュで、透明な部分、半透明な部分あるいは不透明な部分を分けたい場合があります。
フレームバッファアルファのみで画一的にやってしまうと、全てが半透明になってしまい、部分分けができません。

　このサンプルのメッシュは単純な正6面体ですが、中心部分を半透明にしています。言い換えると中心部分のサーフェイス（マテリアル）と外枠部分を分けて、中心部分のアルファを半透明にしています。

　これを反映するには、レンダリングステートの引数にD3DRS_SRCBLEND, D3DBLEND_SRCALPHA及びD3DBLEND_INVSRCALPHAを指定し、さらにマテリアルをブレンドファクターにするよう明示的にSetRenderState(D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_MATERIAL)を指定します。

▼実行結果



```
//アルファブレンディングを設定する
pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
pDevice->SetRenderState ( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
pDevice->SetRenderState ( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
pDevice->SetRenderState( D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_MATERIAL );
```

One Point

　“透明なメッシュ”とは、透明なマテリアルであるメッシュのことです。透明なマテリアルとは、アルファ値が0より大きく1.0未満のマテリアルです。

　マテリアルが透明でないメッシュは通常どおり不透明でレンダリングされます。

181 アルファ付きテクスチャの特定領域だけを透過させる

●テクスチャベースでアルファ値を指定する

中級

サンプルプロジェクト　181　アルファ付きテクスチャを利用

キャプチャ絵を見ると、ボックスの中心部分が透けているのが分かります。

しかし、メッシュの形状自体は単純な6面体です。また、マテリアルも中心と外枠を分けているわけではありません。この透明部分はテクスチャで生成しています。

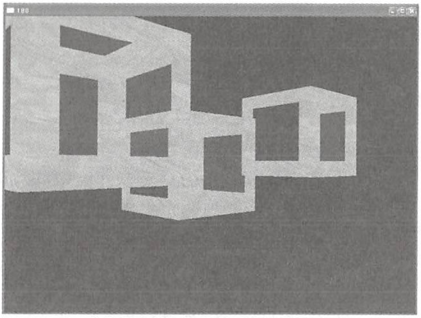
テクスチャ画像を見てみましょう。tga（タルガ）32bit形式です。一見すると真ん中に穴が開いているわけではなく、どうしてこれで中に穴が開でしょう？

アルファチャンネルの部分を見てください。中に穴があくようなアルファマスクが確認できるでしょう。絵自体に穴があいているのではなく、隠れたアルファチャンルで穴を開けて（透明度の違いを表現して）いたのです。

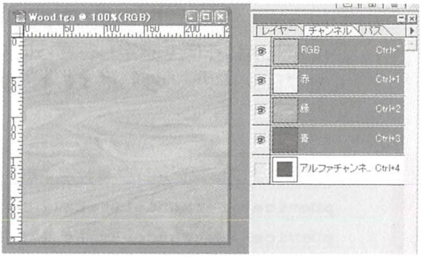
Tips178でも述べましたが、テクスチャアルファの設定はSetTextureStageStateで行います。

このサンプルでは次のように設定しました。

▼実行結果



▼SetTextureStageState



```
//アルファブレンディングを設定する
pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
pDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );

pDevice->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
pDevice->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_DIFFUSE);
pDevice->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);
pDevice->SetTextureStageState(0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
pDevice->SetTextureStageState(0, D3DTSS_ALPHAARG2, D3DTA_DIFFUSE);
pDevice->SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_MODULATE);
```

182 透過している裏ポリゴンを自然にする（カリングトリック）

向きの異なるカリングを2パス行うことで透過を自然に見せる

中級

サンプルプロジェクト　182 カリングトリック

アルファブレンドをかけると、どうしても透過裏のポリゴンが見えなくなってしまったり、逆に余計なポリゴンが見えてしまったりします。

182-1の図は（通常の）カリングを行った場合のレンダリング結果です。全体的にアルファブレンドがかかっているものの、ポリゴンの裏表の関係で“注ぎ口”が見えません。

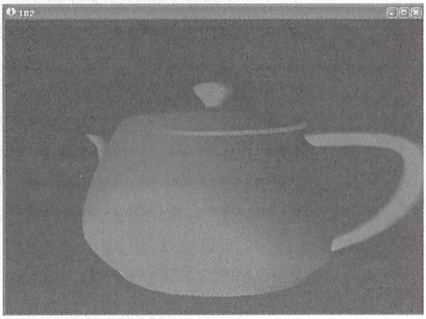
そこで、カリングを行わなければ裏面もレンダリングされるだろうと思い、D3DCULL_NONEフラグに切り替えカリングを無効にしてみると182-2の図のようになります。

注ぎ口はレンダリングされているのですが、それより手前にあるポリゴンまでレンダリングされてしまうので、また見えない状態になってしまいます。さらに、他の余計なポリゴンまでレンダリングされてしまい、汚くなってしまいした。

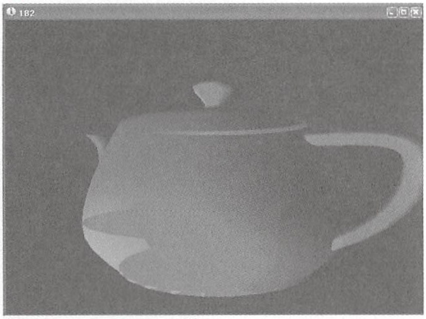
この問題を解決する「カリングトリック」と呼ばれる上手い方法があります。カリングトリックは簡単です。方向が互いに逆な（つまり、裏と表の判断が逆な）カリングを2回行い、それぞれのレンダリングを重ねればいいだけです。Direct3DではD3DCULL_CWとD3DCULL_CCWのフラグのカリングを2回行うことになります。

182-3の図は、カリングトリックによりレンダリングしたものです。問題は全て解消されています。

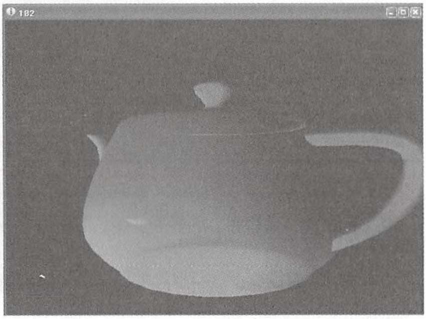
▼実行結果（182-1　通常のカリング）



▼実行結果（182-2　カリング無効）



▼実行結果（182-3　カリングトリック）



```
if( SUCCEEDED( pDevice->BeginScene() ) )
{
    //アルファブレンディングを設定する
    pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );
    pDevice->SetRenderState ( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
    pDevice->SetRenderState ( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
    pDevice->SetRenderState( D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_MATERIAL );

    //ティーポットのレンダリング
    pDevice->SetMaterial(pTeapotMeshMaterials);

    //自然な透過にするため、カリング・トリックを行う
    pDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_CW );
    pTeapotMesh->DrawSubset(0);
    pDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_CCW );
    pTeapotMesh->DrawSubset(0);

    pDevice->EndScene();
}
```

One Point

1つのみのカリングを行った場合、およびカリングを行わない場合もコードを変更して試してみてください。

なお、2回レンダリングしているので処理速度が気になるかもしれませんが、それぞれのレンダリングではポリゴンの片面のみなので、基本的には2回でもカリングオフ時の1回のレンダリングと同じ処理時間になります。

183 情報確認用の文字列を描画する

●ID3DXFontにより文字列を描画する

初級

サンプルプロジェクト 183 フォント描画

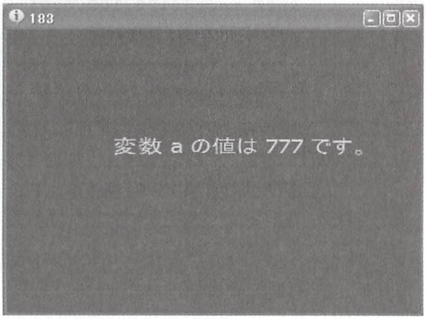
Direct3Dは文字（列）の表示を行うことができます。動的な文字列を描画できるので、変数の値など、プログラム実行中にしか分からない情報の表示が行えます。

サンプルでは、意味もなく変数aに777という値を入れ、変数aの値を表示しています。なんらかのインスタンスの値を見るにはデバッガーを起動しないとなりませんが（デバッグしなければなりなせんが）、いちいちデバッガーを起動するのが面倒な時に重宝します。

フォントを描画するには、ID3DXFontインターフェイスを使用します。

まずはID3DXFontのポインターを宣言します。

▼実行結果



変数 a の値は 777 です。

```
LPD3DXFONT  pFont  =  NULL;
```

それを初期化します。サンプルでは次のような設定にしましたが、サイズ、フォント属性、フォントの種類によって様々でしょう。フォントは適当にtahomaにしました（筆者はtahomaが好きなので）。

なお、フォント名を手に入れる方法は次のTips184で解説しています。

```
//文字列レンダリングの初期化
if(FAILED(D3DXCreateFont(pDevice,0,10,FW_REGULAR,      NULL,
FALSE,SHIFTJIS_CHARSET,
    OUT_DEFAULT_PRECIS,PROOF_QUALITY,FIXED_PITCH | FF_MODERN, "tahoma",
&pFont)))
{
    return E_FAIL;
}
```

初期化が完了すれば、DrawTextメソッドで描画できます。

ポイントとしては、文字列が占める画面上の矩形サイズを取得するために最初にDrawTextを1回実行し、そのあとで本当の描画を行うためにまたDrawTextを実行するという2段階になっているところでしょうか。

```
if(SUCCEEDED(pDevice->BeginScene()))
{
    //試しに変数 a  の値を表示する
    INT a=777;
    TCHAR Str[100];
    sprintf(Str,"変数 a の値は %d です。",a);
    RECT rect={100,100,0,0};
    //文字列のサイズを計算
    pFont->DrawText(NULL,Str,-1,&rect,DT_CALCRECT,NULL);
    // そのサイズでレンダリング
    pFont->DrawText(NULL,Str,-1,&rect,DT_LEFT | DT_BOTTOM,0xff00ff00);

    pDevice->EndScene();
}
```

One Point

ID3DXFontは便利ではありますが、レンダリングコストがかなり高いのでデバッグ目的以外で使用するべきではありません。拙書『はじめての3Dゲーム開発』（工学社）で文字列の高速レンダリングエンジンをソース付きで解説しています。

184 フォント名の一覧ってないの？

●メモ帳で任意のフォント名を取得する

初級

ID3DXFontを初期化する際、フォント名の一覧表が欲しいと思うはずです。誰もが持っているメモ帳（ノートパッド）を利用する、我ながら上手い方法を見つけましたので紹介します。

まず、デスクトップにでも新規テキストドキュメントを作成するなりして、とにかくメモ帳を起動します。

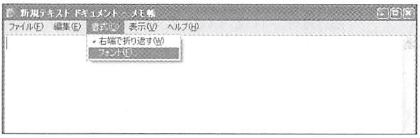
「書式」メニューの「フォント」をクリックします。

すると図のようなダイアログがでるので好きなフォントを選択しましょう。「フォント名（F)」のボックスからコピーできるのでそれをD3DXCreateFontの第11引数（フォント名）にペーストすればいいので簡単です。

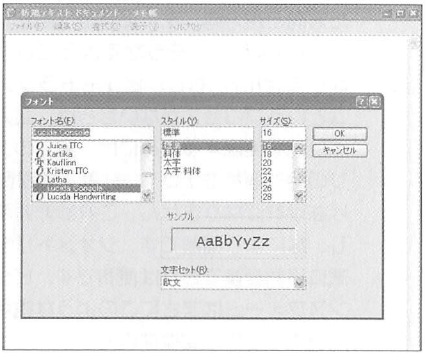
この方法だとダイアログにはPCにインストールされているトゥルータイプフォント（左にTTが付いている）やオープンタイプフォント（左にOが付いている）しか表示されないので確実ですし、フォントのサンプルも表示されるのでそのフォントがどのような雰囲気なのかも一発で分かります。

自分のPCにはあるフォントでも他の人のPCにもあるとは限りません。しかし、その場合は最も近いフォントが適用されるので、さほどナーバスになる必要はありません。

▼「書式」メニューの「フォント」


新規テキスト ドキュメント - メモ帳
ファイル(F)
編集(E)
書式(O)
表示(V)
ヘルプ(H)
✓ 右端で折り返す(W)
フォント(F)...

▼フォント一覧


新規テキスト ドキュメント - メモ帳
ファイル(F)
編集(E)
書式(O)
表示(V)
ヘルプ(H)
フォント
OK
キャンセル
サンプル
AaBbYyZz

フォント名(F): Lucida Console
Juice ITC
Kartika
Kauflinn
Kristen ITC
Latha
Lucida Console
Lucida Handwriting

スタイル(Y): 標準
標準
斜体
太字
太字 斜体

サイズ(S): 16
16
18
20
22
24
26
28

文字セット(R): 欧文

カメラワーク

185 カメラとは

●ビュートランスフォームを理解する

初級

特にカメラとしてのインターフェイスが用意されているわけではありませんが、現実の“カメラ”として捉えると都合がいい事柄があります。それは主にビュートランスフォームです（プロジェクショントランスフォームもカメラの性質に多少関りますが）。

ビュートランスフォームというプロセスがあるお陰で、例えば一人称シューティングゲームで主人公の視点変化を簡単に実現することができます。

どういうことなのかというと、例えば、主人公が右を向いたとします。この時主人公から見れば（視界にある）全ジオメトリは向いた方向と逆方向（左）に移動（回転）します（と言うよりも、そうなるようにしなくてはなりません）。もし、ビュートランスフォームというプロセスがないならば、主人公の視点変化を実現するためには、全てのジオメトリの絶対座標を1つ1つ計算して移動してあげなければなりません。これは非効率的ですし、なにより面倒です。ジオメトリ全体を一気に移動変換できれば便利です。ビュートランスフォームはまさにこのような視点変化を実現するための変換なのです。

比喩的に説明すると、主人公が4つのリンゴジオメトリを真っ直ぐ見ていることとします。リンゴジオメトリは机という絶対座標上に置かれています。主人公が右を向いたとしましょう。主人公から見たリンゴ達は相対的に左に移動（回転）したように見えなくてはなりません。その時、もしビュートランスフォームのような変換がなければ、リンゴジオメトリー個一個を左に回転しなくてはなりません。それよりも、“机ごと”動かせば、机とその上にあるリンゴは全て一斉に動きます。

この机（絶対座標系）ごと動かずという座標変換がビュートランスフォームであり、そしてビュートランスフォームのパラメーターが“カメラ”の特徴になります。

ビュートランスフォームを行う際、視点、注視点、（それと上方ベクトル）を指定しますが、これがそのままカメラの特徴になります。具体的に言うとカメラとは視点と注視点を合わせたものと言えます。

カメラの位置を変える時は視点を移動します。カメラを回転させる場合は、視点をそのままで注視点を移動したり、視点と注視点を回転させたり、何れにしても視点と注視点を結ぶベクトルを回転させます。カメラと呼ばず、“視点”と呼ぶ人もいます。どちらでも構いません。

なお、プロジェクショントランスフォームはカメラそのものではなく“カメラのレンズ“を特徴付けます。

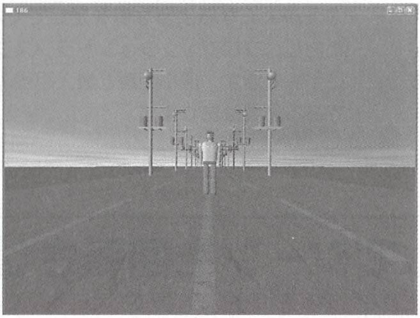
186 カメラを移動してシーンを移動させる

●視点と注視点を同様に移動してシーンに動きを与える

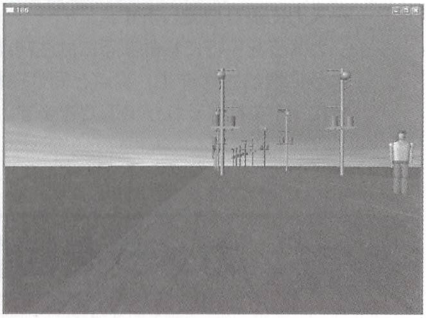
初級

サンプルプロジェクト　186 カメラの移動

▼実行前



▼実行結果



カメラを平行移動します。サンプルでは横移動だけですが、縦の移動もyを増減されればいいだけです。

```
case WM_KEYDOWN:
switch((CHAR)wParam)
{
    case VK_ESCAPE:
          PostQuitMessage(0);
    break;
    case VK_LEFT:
          x-=1;
          break;
    case VK_RIGHT:
          x+=1;
          break;
}

//カメラワーク
 //カメラの単純移動
 {
     // ビュートランスフォーム（視点座標変換）
     D3DXVECTOR3 vecEyePt( x, y, z);  //カメラ（視点）位置
```

```
        D3DXVECTOR3 vecLookatPt(x, y, z+1);//注視位置
        D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
        D3DXMATRIXA16 matView;
        D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
        pDevice->SetTransform(D3DTS_VIEW, &matView);
    }
```

カメラの動き（回転等を含む）はビュートランスフォームでの書き方で反映されます。vecEyePtがカメラの位置です。x成分はキー入力により増減します。

カメラを平行に移動するには注視点も同じように増加させます。視点と同じ座標のz成分に＋1すればそれが実現できます。もし＋1しないと視点とピッタリ重なってしまい画面に何も映らなくなります。

なお＋0.1でも＋10でも構いません。見栄えがよくなるような値を足してください。ただし、プロジェクショントランスフォームに引数で渡す、最小距離と最大距離の範囲内である必要はあります。

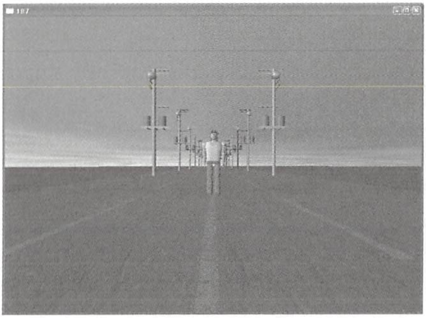
187 カメラを回転してシーンを回転させる

●視点と注視点を同様に回転してシーンに動きを与える

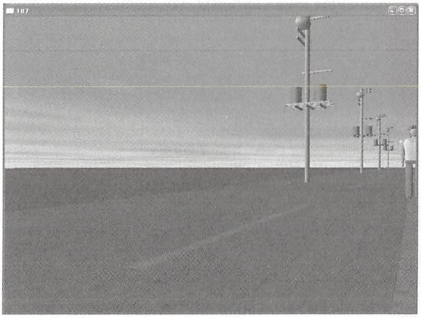
初級

サンプルプロジェクト　187 カメラの回転

▼実行前



▼実行結果



カメラの回転はビュートランスフォームでの書き方で反映されます。回転させる方法は1つではありませんが、ここでは注視点を回転させることによりカメラを回転させます（回転の中心は視点vecEyePtです）。

回転させるために回転行列を作成します、回転量はキー入力により増減する変数Angleです。

ポイントは、回転させる前の座標を0,0,1として、それを回転させます。後で始点の位置

ベクトルを足すことにより、結果的に視点を中心に回ることとなります。

ここで注意しなければならないことは、最初から注視点を（x,y,z+1）などとして、それを回せば一手間減らせると思ってしまうことです。それだと（x,y,z+1）を絶対座標軸まわりの回転をしてしまうので、大きく回ってしまい、とんでもない位置にいってしまうことになります。

なお、z成分に足す値は+0.1でも+10でも構いません。見栄えがよくなるような値を足してください。ただし、プロジェクショントランスフォームに引数で渡す、最小距離と最大距離の範囲内である必要はあります。

```
case WM_KEYDOWN:
switch((CHAR)wParam)
{
   case VK_ESCAPE:
        PostQuitMessage(0);
   break;
   case VK_LEFT:
        Angle-=0.1;
        break;
   case VK_RIGHT:
        Angle+=0.1;
        break;
}

//カメラの回転
{
    // ビュートランスフォーム（視点座標変換）
    D3DXVECTOR3 vecEyePt( x, y, z);  //カメラ（視点）位置
    D3DXVECTOR3 vecLookatPt(0 , 0, 1);//注視位置
    D3DXMATRIX mat;
    D3DXMatrixRotationY(&mat,Angle);
    D3DXVec3TransformCoord(&vecLookatPt,&vecLookatPt,&mat);
    vecLookatPt+=vecEyePt;
    D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
    D3DXMATRIXA16 matView;
    D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
    pDevice->SetTransform(D3DTS_VIEW, &matView);
 }
```

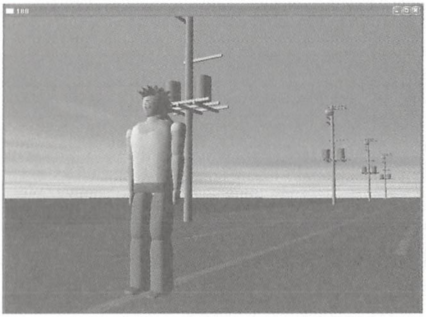
188 特定の物体を注視させる

●注視点を被写体に固定して注視させる

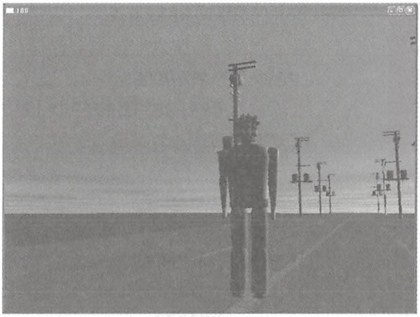
初級

サンプルプロジェクト　188 物体を注視

▼実行結果



▼実行結果

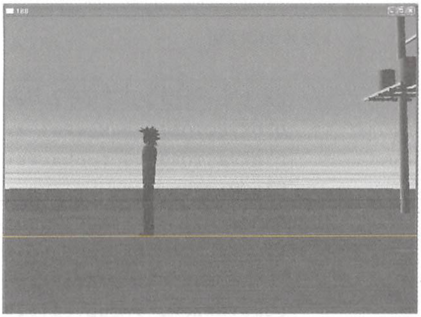


実は被写体を注視させることは最も簡単です。単に、注視点を被写体の位置にすればいいだけです。

Thing[1]は被写体（人間メッシュ）のコンテナで、位置ベクトルvecPositionを保持しています。

それを単純に注視点にしています。3枚の図は、カメラがどのような位置にあろうとも。全て被写体を中心に捉えています。

▼実行結果



```
//被写体を常に注視
 {
     // ビュートランスフォーム（視点座標変換）
     D3DXVECTOR3 vecEyePt( x, y, z);  //カメラ（視点）位置
     D3DXVECTOR3 vecLookatPt(Thing[1].vecPosition.x, Thing[1].
            vecPosition.y, Thing[1].vecPosition.z+1);//注視位置
     D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
     D3DXMATRIXA16 matView;
     D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
     pDevice->SetTransform(D3DTS_VIEW, &matView);
 }
```

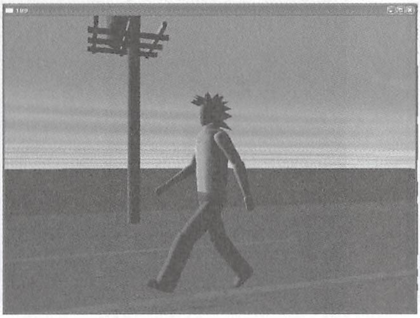
189 フロートラックを行う（注視＋追従移動）

カメラを台車に乗せたような撮影を行う

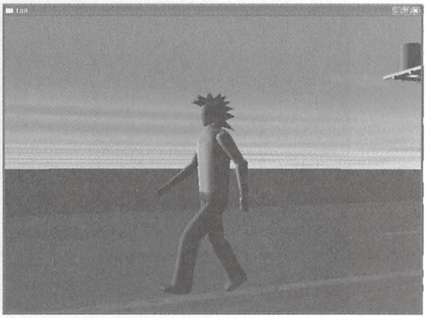
初級

サンプルプロジェクト 189 フォロートラック

▼実行結果



▼実行結果



被写体の移動速度にシンクロして移動るような台車にカメラを乗せることにより、被写体を追従するような撮影方法は「フォロートラック」と呼ばれます。

3DCGの場合、台車に乗せるまでもなくそのままカメラを動かせば済みます。もともと3DCGのカメラの動きは手ぶれなどが無くリニアに動くので、そもそも台車に乗せているようなものです。逆に手振れなどリアルな映し方のほうが難しいのですから。

実装は簡単です。

ここでは次のように被写体の動きにシンクロさせています。

```
//フォロートラック撮影
 {
     //被写体を動かす
     Thing[1].vecPosition.z-=0.006;
     // ビュートランスフォーム（視点座標変換）
     D3DXVECTOR3 vecEyePt=Thing[1].vecPosition+D3DXVECTOR3(3,0,-1);//カメラ（視点）位置を被写体の横3単位、後ろ1単位の場所に置く
     D3DXVECTOR3 vecLookatPt=Thing[1].vecPosition;//注視位置は被写体の座標そのまま　被写体を注視させる
     D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
     D3DXMATRIXA16 matView;
     D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
     pDevice->SetTransform(D3DTS_VIEW, &matView);
 }
```

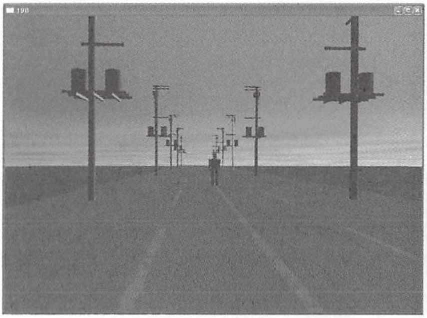
190 回り込みトラックを行う（注視＋移動＋回転）

●カメラを台車に乗せたような撮影を行う

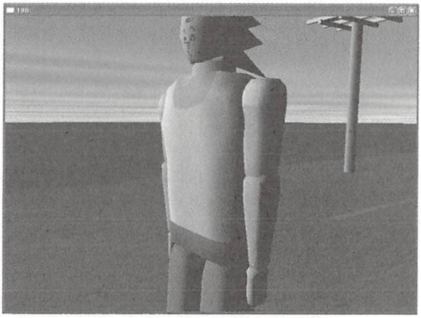
初級

サンプルプロジェクト　190 回り込みトラック

▼実行結果



▼実行結果



直前のTipsと同じく回り込みトラックも台車にカメラを乗せる撮影方法です。

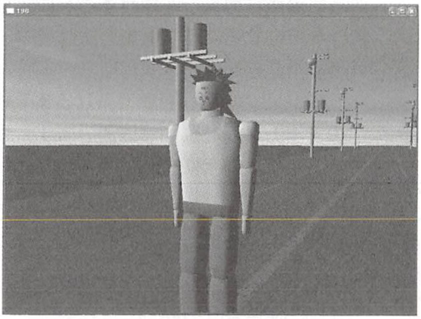
ただ回り込みトラックは、被写体の動きにシンクロするのではなく、被写体よりも（相対）速度を速くします。

つまり、被写体が止まっている時、あるいは動いている時でも、最初は離れた場所にあるカメラをだんだんと被写体に近づけます。カメラは常に被写体を注視します。

これも実装は簡単です。

ここでは次のように被写体に注視させ、かつ、カメラをだんだんと近づけています。

▼実行結果



```
//回り込みトラック撮影
 {
        //被写体の近くにいくまでカメラを動かす
        if(vecCameraPos.z > Thing[1].vecPosition.z-2)
        {
                vecCameraPos.z-=0.1;
        }
        // ビュートランスフォーム（視点座標変換）
```

```
    D3DXVECTOR3 vecEyePt=vecCameraPos;  //カメラ（視点）位置
    D3DXVECTOR3 vecLookatPt(Thing[1].vecPosition.x,Thing[1].vecPosition.y,
        Thing[1].vecPosition.z);//注視位置
    D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
    D3DXMATRIXA16 matView;
    D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
    pDevice->SetTransform(D3DTS_VIEW, &matView);
}
```

vecCameraPosはカメラの位置を保持するグローバルなベクトルです。

毎フレーム自動的にカメラを移動します。視点はvecCameraPosをそのまま使用し、そして、カメラがどの位置にあっても被写体（人間メッシュ）を注視するように注視点は人間メッシュの位置をそのまま使っています。

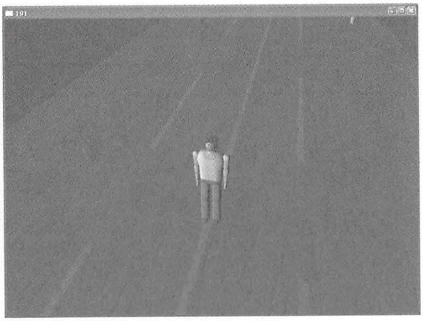
191 クレーンを行う（注視＋移動）

●カメラをクレーンに乗せたような撮影を行う

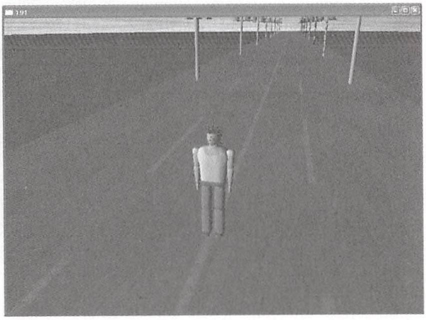
初級

サンプルプロジェクト　191 クレーン

▼実行結果



▼実行結果



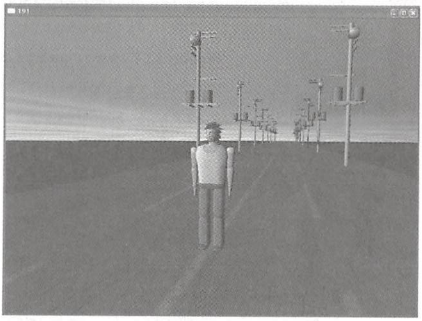
「クレーン」は、文字通りの撮影方法です、つまりクレーン（クレーン車）にカメラを乗せて、ブレのないリニアな絵を撮影する方法です。フォロートラックのTipsでも述べましたが、3DCGはもともとブレのないカメラワークが得意ですから（というかブレを出すほうが手間がかかります）、特別なコーディングは一切不要です。

実装はあまりにも簡単、単純すぎるので解説することはないでしょう。

単にカメラ位置のy成分を、だんだんとマイナスしているだけのことです。もちろん、

被写体を捉え続けるように、人間メッシュの座標を注視点にしています。

▼実行結果



```
//クレーン撮影
 {
        //クレーン上にある（とイメージした）カメラを平行移動する
        if(y> 1)
        {
                y-=0.003;
        }
        // ビュートランスフォーム（視点座標変換）
        D3DXVECTOR3 vecEyePt( x, y, z);  //カメラ（視点）位置
        D3DXVECTOR3 vecLookatPt(Thing[1].vecPosition.x,Thing[1].vecPosition.y,
             Thing[1].vecPosition.z);//注視位置
        D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
        D3DXMATRIXA16 matView;
        D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
        pDevice->SetTransform(D3DTS_VIEW, &matView);
 }
```

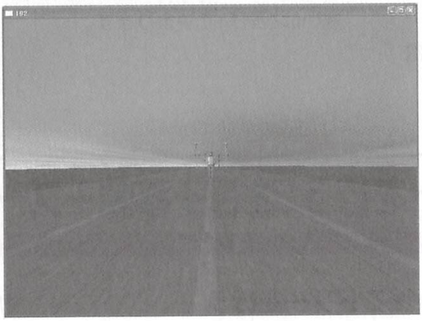
192 ズームイン・アウトする

●プロジェクショントランスフォームの視野角を変換させてズーム効果を出す

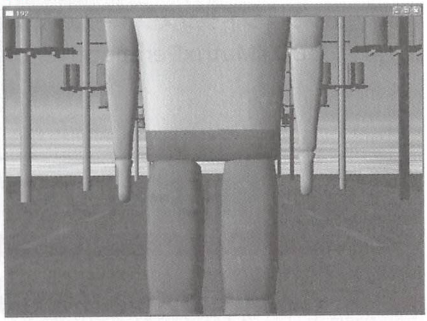
初級

サンプルプロジェクト　192 ズームイン・ズームアウト

▼実行結果（ズームアウト）



▼実行結果（ズームイン）



2枚の図を比べてパースのかかり具合が大分異なるのが分かるでしょうか？

被写体（人間メッシュ）の大きさが異なっていますが、これはカメラの位置を移動したのではありません。被写体、カメラ共に同じ位置です。大きさが異なるのはズームによるものです。言い方を変えればレンズの違いです（レンズの効果です）。それが、2枚のパースが異なっている理由です。

```
FLOAT Zoom=4;

case WM_KEYDOWN:
    switch((CHAR)wParam)
    {
          case VK_ESCAPE:
                PostQuitMessage(0);
          break;
          case VK_LEFT:
                Zoom+=0.1;
                break;
          case VK_RIGHT:
                Zoom-=0.1;
                break;
    }
```

```
//ズームイン。ズームアウト
 {
      // プロジェクショントランスフォーム（射影変換）
      D3DXMATRIXA16 matProj;
      D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI/Zoom,
(FLOAT)WINDOW_WIDTH/(FLOAT)WINDOW_HEIGHT, 0.1f, 1000.0f);
      pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
 }
```

ズーム、あるいはパースの違いは、プロジェクショントランスフォームの反映です。

D3DXMatrixPerspectiveFovLH関数の第2引数は視野角の引数であり、ズームの違いは視野角で決まります。

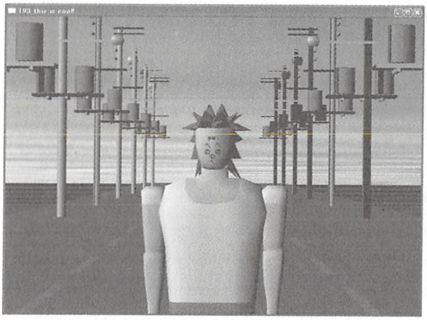
193 ドリーイン・アウトを行う（注視＋移動＋ズーム）

●ズーム効果とカメラ移動を組み合わせてドリーを行う

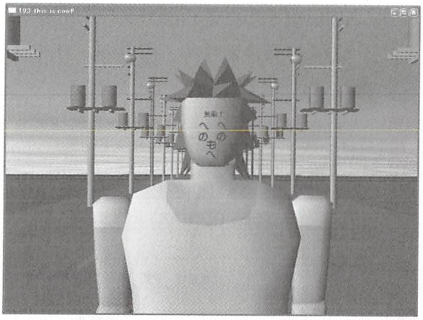
初級

サンプルプロジェクト　193 ドリーイン・ドリーアウト

▼実行結果



▼実行結果



「ドリーイン」(dolly in)、「ドリーアウト」(dolly out)のdollyはカメラを乗せる台車のことですが、フォロートラックとは異なるカメラワークです。ドリーは、映画などでよく“ビックリ”した時や“愕然・呆然”とした時に使われるカメラワークです。ドリーインはカメラ自体が被写体に近づき、同時に画角を変化させて被写体の映る大きさを変化させないようにする技法です。

上手いドリーインでは被写体の画面上の大きさが変わらず、背景だけがズームアウトされているように見えます。

ドリーアウトはドリーインの逆です。

サンプルでは、左矢印キーで「ドリーイン」、右矢印キーで「ドリーアウト」します。サンプルは、被写体の大きさはけっこう変化しますので、あまり上手いドリーとは言えません。カメラの視野角を単純に線形で変化させているからです。もっと本格的なドリーを行うとすれば、カメラと被写体の距離に応じて視野角の“変化量”を変化させるようにすることになるでしょうが、面倒なので線形にしました。

```
FLOAT x=0,y=1.7,z=-100;
FLOAT Pers=D3DX_PI/24;

switch((CHAR)wParam)
{
    case VK_ESCAPE:
          PostQuitMessage(0);
    break;
    case VK_LEFT:
          if(z < Thing[1].vecPosition.z-2)
          {
                z+=0.1;
                Pers+=0.002;
          }
          break;
    case VK_RIGHT:
          if(z > Thing[1].vecPosition.z-10)
          {
                z-=0.1;
                Pers-=0.002;
          }
          break;
}


//ドリーイン・ドリーアウト　映画で良く見る”びっくり”した時のカメラワーク
 {
     // ビュートランスフォーム（視点座標変換）
     D3DXVECTOR3 vecEyePt( x, y, z);  //カメラ（視点）位置
     D3DXVECTOR3 vecLookatPt(x, y, z+1);//注視位置
     D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
     D3DXMATRIXA16 matView;
     D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
     pDevice->SetTransform(D3DTS_VIEW, &matView);
     // プロジェクショントランスフォーム（射影変換）
     D3DXMATRIXA16 matProj;
```

```
        D3DXMatrixPerspectiveFovLH(&matProj, Pers, (FLOAT)WINDOW_WIDTH/(FLOAT)
WINDOW_HEIGHT, 0.1f, 1000.0f);
        pDevice->SetTransform(D3DTS_PROJECTION, &matProj);
  }
```

ドリーは、カメラの移動とレンズの倍率の両方が同時に変化するので、ビュートランスフォームとプロジェクショントランスフォームの2つを同時に調整することになります。

194 カメラを画面に対して絶対的に回転させる

●姿勢行列を保持して画面に対して常に一定の方向で回転させる

中級

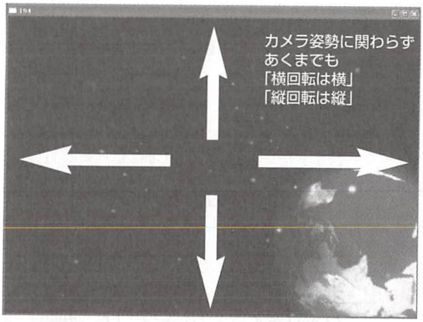
サンプルプロジェクト　194 絶対的な回転

なにも工夫せずにD3DXMatrixRotation関数を組み合わせた回転をそのままさせると、ヨー回転とピッチ回転が混ざった回転であったり、ヨー回転をさせているのに画面に対してピッチ回転をしたりしてしまうことになります。これでは意図した姿勢にするための操作は不可能であるとも言えます。

今画面に映っているシーンに対して、言い方を変えれば現在のビューに対して“絶対的な”ヨー回転あるいはピッチ、ロール回転をして欲しい場合は多いと思います。

これを実現するには、現在の姿勢を行列に保持し、フレーム毎の回転は直前のフレームからの“増分”のみをかけると上手くいきます。このように姿勢を保持する行列を姿勢行列と言います。

▼実行結果



```
FLOAT fYaw=0,fPitch=0,fRoll=0;
D3DXMATRIX g_matRotation;
g_matRotationが姿勢を保存する姿勢行列です。

case WM_KEYDOWN:
switch((CHAR)wParam)
{
    case VK_ESCAPE:
           PostQuitMessage(0);
```

```
        return 0;
    case VK_RIGHT:
          fYaw=0.05f;
          break;
    case VK_LEFT:
          fYaw=-0.05f;
          break;
    case VK_DOWN:
          fPitch=0.05f;
          break;
    case VK_UP:
          fPitch=-0.05f;
          break;
    case 'X':
          fRoll=0.05f;
          break;
    case 'Z':
          fRoll=-0.05f;
          break;
}

// ビュートランスフォーム（ワールド座標→カメラ座標への変換）
CurrentAttitude(&g_matRotation,fYaw,fPitch,fRoll);//保存されている現在姿勢を出発点として回転させる
fYaw=fPitch=fRoll=0;
D3DXVECTOR3 vecEye(0,0,0);
D3DXVECTOR3 vecLook(0,0,1);
D3DXVECTOR3 vecUp(0,1,0);
D3DXVec3TransformCoord(&vecLook,&vecLook,&g_matRotation);
D3DXVec3TransformCoord(&vecUp,&vecUp,&g_matRotation);
D3DXMatrixLookAtLH(&matView,&vecEye,&vecLook,&vecUp);
pDevice->SetTransform(D3DTS_VIEW, &matView);
```

●CurrentAttitude関数　毎フレームの増分姿勢を姿勢行列に蓄積する

```
HRESULT CurrentAttitude(D3DXMATRIX* pmatAttitude,FLOAT fYaw,FLOAT fPitch,FLOAT fRoll)
{
    static BOOL  boOneTime=FALSE;
    if(!boOneTime)
    {
          boOneTime=TRUE;
          D3DXMatrixIdentity(pmatAttitude);
    }
    D3DXMATRIX matAxis;
    if(fYaw !=0)
    {
          D3DXMatrixRotationAxis(&matAxis,&D3DXVECTOR3(pmatAttitude-
>_21,pmatAttitude->_22,pmatAttitude->_23),fYaw);
    }
    else if(fPitch !=0)
    {
          D3DXMatrixRotationAxis(&matAxis,&D3DXVECTOR3(pmatAttitude-
>_11,pmatAttitude->_12,pmatAttitude->_13),fPitch);
    }
    else if(fRoll !=0)
    {
          D3DXMatrixRotationAxis(&matAxis,&D3DXVECTOR3(pmatAttitude-
>_31,pmatAttitude->_32,pmatAttitude->_33),fRoll);
    }
    else
    {
          return S_FALSE;
    }
    *pmatAttitude*=matAxis;

    return S_OK;
}
```

195 カメラを画面に対して絶対的に回転させる②（クォータニオン版）

●クォータニオンに回転を保持して画面に対して常に一定の方向で回転させる　中級

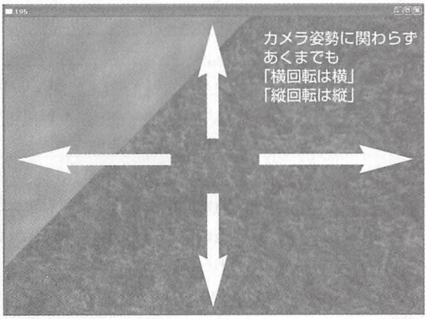
サンプルプロジェクト　195　絶対的な回転その2

直前のサンプルでは行列に姿勢を保持しましたが、ここではクォータニオンに姿勢を保持します。直前のサンプルとの違いはそれだけです。

ただ、クォータニオンを使用することによりシンプルになっています。姿勢行列の時のようなCurrentAttitude関数などの関数が不用になっています。

ポイント部分は、ウィドウプロシージャ内における姿勢クォータニオンの更新部分です。それぞれのキーに対応した回転の増分を姿勢クォータニオンに蓄積しています。

▼実行結果



```
case WM_KEYDOWN:
switch((CHAR)wParam)
{
    case VK_ESCAPE:
        PostQuitMessage(0);
    return 0;
    // Qをクォータニオン、vを軸ベクトル、θを回転角とすると Q=cos(1/2θ) +sin(1/2θ)*V
    //D3DXQUATERNIONのwは実部、x,y,zは虚部の実数（回転軸ベクトルとして振舞う、但し、単位ベクトルである必要がある）
    case VK_RIGHT:
        //sin(1/2θ)*(0,1,0)
        qtnDelta.x=0;
        qtnDelta.y=sin(-DELTA_ANGLE/2.0f);
        qtnDelta.z=0;
        qtnDelta.w=cos(-DELTA_ANGLE/2.0f);
        qtnAttitude*=qtnDelta;
        break;
    case VK_LEFT:
        //sin(1/2θ)*(0,1,0)
        qtnDelta.x=0;
        qtnDelta.y=sin(DELTA_ANGLE/2.0f);
        qtnDelta.z=0;
        qtnDelta.w=cos(DELTA_ANGLE/2.0f);
        qtnAttitude*=qtnDelta;
        break;
    case VK_DOWN:
        //sin(1/2θ)*(1,0,0)
        qtnDelta.x=sin(-DELTA_ANGLE/2.0f);
        qtnDelta.y=0;
        qtnDelta.z=0;
        qtnDelta.w=cos(-DELTA_ANGLE/2.0f);
```

```
          qtnAttitude＊=qtnDelta;
          break;
    case VK_UP:
          //sin(1/2θ)＊(1,0,0)
          qtnDelta.x=sin(DELTA_ANGLE/2.0f);
          qtnDelta.y=0;
          qtnDelta.z=0;
          qtnDelta.w=cos(DELTA_ANGLE/2.0f);
          qtnAttitude＊=qtnDelta;
          break;
    case 'X':
                //sin(1/2θ)＊(0,0,1)
          qtnDelta.x=0;
          qtnDelta.y=0;
          qtnDelta.z=sin(-DELTA_ANGLE/2.0f);
          qtnDelta.w=cos(-DELTA_ANGLE/2.0f);
          qtnAttitude＊=qtnDelta;
          break;
    case 'Z':
                //sin(1/2θ)＊(0,0,1)
          qtnDelta.x=0;
          qtnDelta.y=0;
          qtnDelta.z=sin(DELTA_ANGLE/2.0f);
          qtnDelta.w=cos(-DELTA_ANGLE/2.0f);
          qtnAttitude＊=qtnDelta;
          break;
}
```

ここではD3DX系の算術関数を用いず、手作業でクォータニオンを計算していますので、仕組みを把握しやすいでしょう。計算式はコメントに書いてあるとおりの式を適用します。

あとはD3DXMatrixRotationQuaternion関数により、その姿勢クォータニオンでビュートランスフォームするだけです。

```
// ビュートランスフォーム（ワールド座標→カメラ座標への変換
{
    //クォータニオン（qtnAttitude）を回転量パラメーターに使用する
    {
          D3DXMatrixRotationQuaternion(&matView,&qtnAttitude);
    }
    pDevice->SetTransform(D3DTS_VIEW, &matView);
}
```

196　カメラに対し一定なジオメトリを作る（ビルボーディング）

●ビュー変換の打ち消し行列を作成する

中級

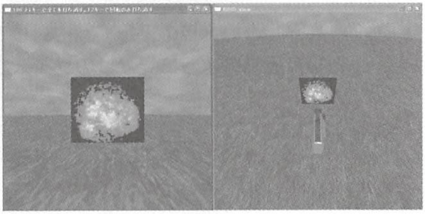
サンプルプロジェクト　196 ビルボーディング

ビュー変換の一部あるいは全部を打ち消すことにより、特定のジオメトリを常にカメラに向けたり、あるいは常に画面の固定位置に留めさせる処理はビルボーディングと呼ばれ、ビルボーディングがかかった状態のそのジオメトリはビルボードと呼ばれます。

現実世界において、歩行者や車の方向を向くように設置された路上看板や街頭看板などの広告用看板は英語でビルボードと言い、それに似ているからです。

サンプルを起動した直後なにもしない状態において、カメラの初期位置及びその状態でのビューは右の図のようになります。

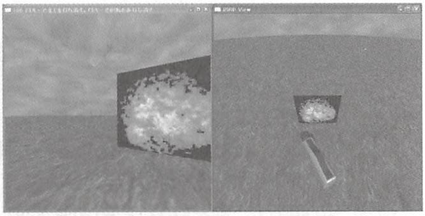
▼カメラ初期状態とそのビュー



●ビルボーディングをかけないで、普通にビュートランスフォームした場合

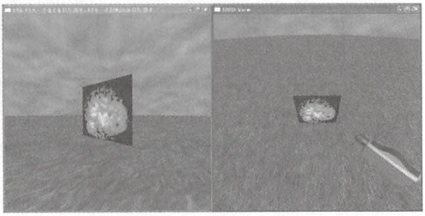
まずカメラを回転させると当然シーン上のジオメトリは一斉に見かけ上回転しますから、Plateメッシュ（爆発絵が貼ってあるメッシュ）も傾斜して見えます。

▼ビルボーディングをかけない時　カメラの回転



そしてカメラを動かすと、これも当然シーンが移動しますからPlateメッシュも大きく見えたり小さく見えたりします。

▼ビルボーディングをかけない時　そして移動



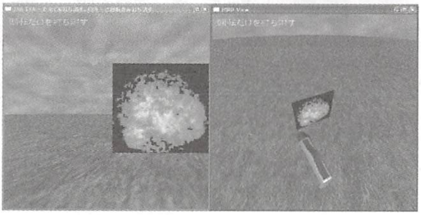
●回転のみ打ち消すビルボーディングをかけた場合

では、ビルボーディングをかけてみましょう。まずは回転のみ打ち消すビルボーディングをかけた時の効果をみてみましょう。

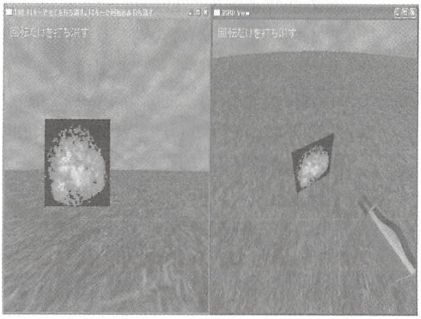
カメラが回転するとPlateメッシュもカメラの回転にシンクロして回転します。その結果、見かけ上はPlateメッシュは回転しません。

しかし、カメラの平行移動に対しては依然として通常のように見かけのサイズが変化します。

▼回転のみ打ち消すようなビルボーディングをかけた時　カメラの回転は影響しない


回転だけを打ち消す
BIRD View
回転だけを打ち消す

▼回転のみ打ち消すようなビルボーディングをかけた時　移動は反映される


回転だけを打ち消す
BIRD View
回転だけを打ち消す

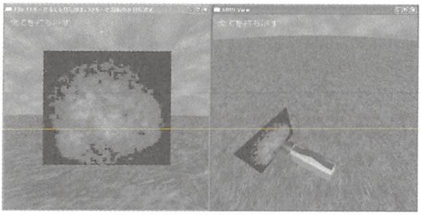
●全てを打ち消すビルボーディングをかけた場合

今度はカメラの姿勢を全て打ち消すビルボーディングをかけてみます。

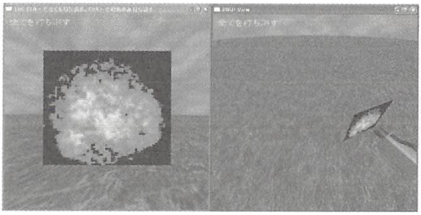
次の2つの図から明らかなように、カメラを回転させようが平行移動しようが画面上のジオメトリはピクともしません。これはカメラの回転にシンクロしてPlateメッシュが回転し、さらにカメラの移動にもシンクロして追従するからであり、言うなればカメラの前にくっつけて固定したかのようになるからです。

なお、ライトの影響までを打ち消すものではないので、ライトの影響は受けます。実際、上図と下図ではライトのかかり具合が異なるので明るさが異なっています。

▼全てを打ち消すビルボーディングをかけた時　回転しても影響なし


全てを打ち消す
BIRD View
全てを打ち消す

▼全てを打ち消すビルボーディングをかけた時　移動も影響しない


全てを打ち消す
BIRD View
全てを打ち消す

One Point

Plateメッシュの絵は、“ふち”が黒くレンダリングされていますが、これは傾斜具合を視認しやすくなるよう敢えてそうしているものです。通常は色抜きにより黒い部分を透明にします。

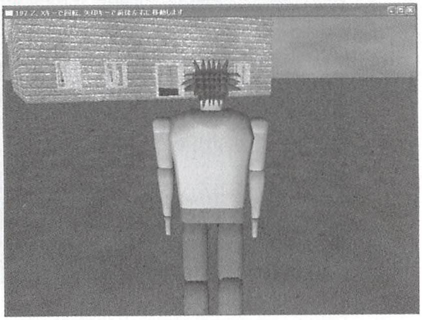
197 3人称視点（TPV）

カメラが主人公を見下ろすようなビューを作成する

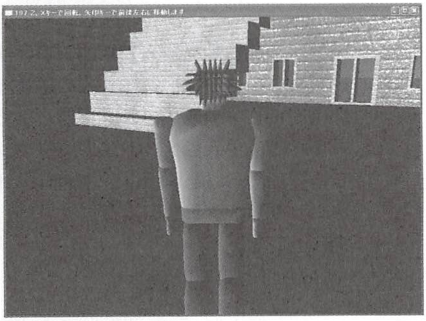
中級

サンプルプロジェクト 197 TPV

▼実行前



▼実行結果



3人称視点（Third Person View）は略してTPVと呼ばれます。TPVは特別なことではなく、単に見下ろしカメラを主人公に追従されることなので、アングルは微妙に異なるもののTips189「フォロートラック」と仕組みは全く同じです。

3人称視点となるビュー変換のコードの書き方は無数にあります。ここでは次のようにしています。

Thnig[3]は主人公のインスタンスです。したがってThing[3].vecPositionは主人公の位置を意味し、Thing[3].matRotationは主人公の姿勢行列を意味します。

```
// ビュートランスフォーム（ワールド座標→カメラ座標への変換）
D3DXMATRIX matView;
D3DXVECTOR3 vecEyePt(0,0.5,-2 );
D3DXVECTOR3 vecLookatPt(0,0,0);
D3DXVec3TransformCoord(&vecEyePt,&vecEyePt,&Thing[3].matRotation);
D3DXVec3TransformCoord(&vecLookatPt,&vecLookatPt,&Thing[3].matRotation);
vecEyePt+=Thing[3].vecPosition;
vecLookatPt+=Thing[3].vecPosition;
```

```
D3DXVECTOR3 vecUpVec(0.0f, 1.0f, 0.0f);//上方位置
D3DXMatrixLookAtLH(&matView, &vecEyePt, &vecLookatPt, &vecUpVec);
pDevice->SetTransform(D3DTS_VIEW, &matView);
```

198 1人称視点(FPV)⇔3人称視点(TPV)相互切り替え

●1、2行の変更で人称を切り替える

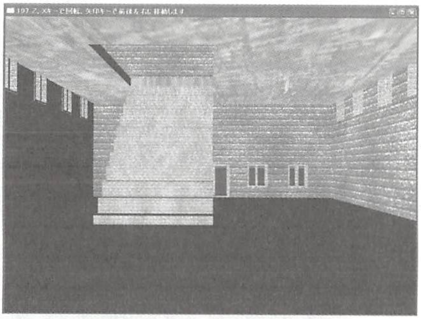
中級

1人称視点と3人称視点は、ほんの1、2行コードを書き換えるだけで相互に変換できます。

カメラの位置を主人公に重ねさえすればいいだけなのですから。主人公の顔あるいは目の位置にカメラを置けば完璧でしょう。

そして、主人公メッシュのポリゴンによって視界が遮られる場合は、主人公のメッシュをレンダリングしないようにすればいいだけのことです。

▼実行結果



282行と283行の

```
D3DXVECTOR3 vecEyePt(0,0.5,-2 );
D3DXVECTOR3 vecLookatPt(0,0,0);
```

を、次のようにすれば

```
D3DXVECTOR3 vecEyePt(0,1.6,0 );
D3DXVECTOR3 vecLookatPt(0,1.6,1);
```

vecEyePtはカメラ位置ですから、それを主人公にぴったり重ねます。y成分を1.6にしているのは、ここで使用している人間メッシュを170cmで作成しているのでだいたい目の位置に配置するためです。

注視点はカメラ位置からz方向に適当な値を入れるようにします。この例では、目から手前1単位のところを見るようにしています。カメラ位置と注視点のy成分が同じなので目線は地面と平行になります。

199 ビュー変換行列を自分で作る

D3DXを利用せず、自前でビュー変換行列を作成する

中級

サンプルプロジェクト　199 ビュー行列の手動作成

ワールド空間からカメラ空間の系へ変更することが、ビュートランスフォームなわけですが、ビュートランスフォーム行列は、ある系から別の系への変換を行う仕組みを把握するのに役立ちます。

ビュー変換行列を作成するということは、視点、注視点、上方ベクトルからカメラ空間における基底軸を作ることということが分かるでしょうか。

次の擬似コードはDirect3Dにおけるビュー変換行列の作成原理を表します。

●擬似コード

```
zaxis = normal(At - Eye)
xaxis = normal(cross(Up, zaxis))
yaxis = cross(zaxis, xaxis)

xaxis.x               yaxis.x              zaxis.x              0
xaxis.y               yaxis.y              zaxis.y              0
xaxis.z               yaxis.z              zaxis.z              0
-dot(xaxis, eye)    -dot(yaxis, eye)    -dot(zaxis, eye)    1
```

これをそのままコーディングしたのが次のコード片です。

サンプルでは、このビュー変換行列によりビュートランスフォームしています。結果はもちろんD3DXMatrixLookAtLHを使用した時と全く同じです。

```
D3DXMATRIXA16 matView;
//ビュー行列を自分で作成
 {
     D3DXVECTOR3 X,Y,Z;
     D3DXVec3Normalize(&Z , &(vecLookatPt - vecEyePt));
     D3DXVec3Normalize(&X , D3DXVec3Cross(&vecUpVec, &vecUpVec, &Z));
     D3DXVec3Cross(&Y, &Z , &X);

     matView._11=X.x;
     matView._21=X.y;
     matView._31=X.z;
     matView._41=-D3DXVec3Dot(&X,&vecEyePt);
```

```
        matView._12=Y.x;
        matView._22=Y.y;
        matView._32=Y.z;
        matView._42=-D3DXVec3Dot(&Y,&vecEyePt);

        matView._13=Z.x;
        matView._23=Z.y;
        matView._33=Z.z;
        matView._43=-D3DXVec3Dot(&Z,&vecEyePt);

        matView._14=0;
        matView._24=0;
        matView._34=0;
        matView._44=1;
}
```

マルチビュー

200　1つのビューを分割した領域にレンダリングする

1つのウィンドウクライアント領域を複数に分割する

中級

サンプルプロジェクト　200 画面分割その1

ここでは、1つのウィンドウを4つの領域に分割し、それぞれの領域に異なる視点からのビューをレンダリングするということを行います。

仕組みとしては、ビューポートを調整することによって、1つのウィンドウのクライアント領域上に複数のビューをレンダリングすることを実現します。ビューポートによってモニター上に描画される領域をビューポート矩形と言います。

通常、ビューポート矩形サイズはクライアント領域のサイズに一致しています、例えば、ウィンドウのサイズが800×600ピクセルであれば、ビューポート矩形も800×600ピクセルとなっています。

ビューポート矩形は、ウィンドウのサイズとは無関係に任意のサイズに設定することができます。したがって、例えば、ウィンドウサイズが800×600ピクセルであったとしても、ビューポート矩形を100×100ピクセルにすることもでき、その場合、レンダリングが描画される部分は100×100ピクセルの領域でしかなく、クライアント領域のほとんどはウィンドウのバックグランドカラーとなることになります。

また、ビューポート矩形は、描画開始位置を任意の位置に設定できます。通常はクライアント座標(0,0)が描画開始点になりますが、

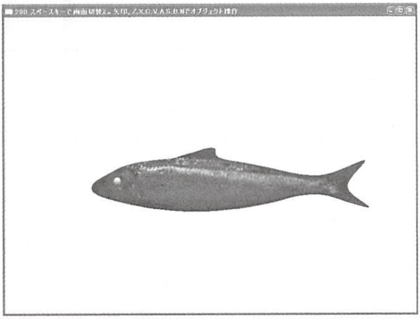
例えば、(400,300)とすれば画面真ん中から描画することができます。

そして、描画開始位置を設定することにより、好きな場所から描画を開始できます。つまり、矩形を移動できます。

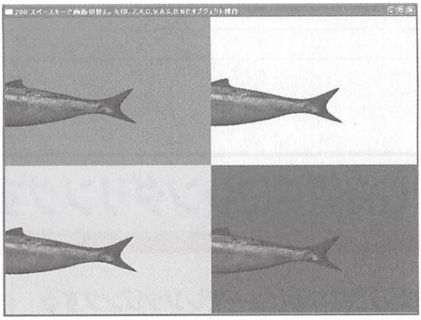
この機能を応用すれば、4画面分割ができるのは想像できるでしょう。

ビューポート矩形のサイズと位置を変えながら4回レンダリングしています。ViewRender関数はなにも特別なことはしていません。通常のレンダリングコードです。

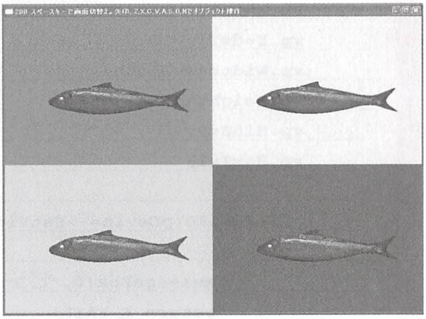
▼実行前


200 スペースキーで画面切替え。矢印、Z,X,C,V,A,S,B,Nでオブジェクト操作

▼実行結果


200 スペースキーで画面切替え。矢印、Z,X,C,V,A,S,B,Nでオブジェクト操作

▼実行結果


200 スペースキーで画面切替え。矢印、Z,X,C,V,A,S,B,Nでオブジェクト操作

```
ChangeViewport(0,0,g_VP.Width/2,g_VP.Height/2);
ViewRender(D3DXCOLOR(1,0,0,1));

ChangeViewport(0,g_VP.Height/2,g_VP.Width/2,g_VP.Height/2);
ViewRender(D3DXCOLOR(0,1,0,1));

ChangeViewport(g_VP.Width/2,g_VP.Height/2,g_VP.Width/2,g_VP.Height/2);
ViewRender(D3DXCOLOR(0,0,1,1));

ChangeViewport(g_VP.Width/2,0,g_VP.Width/2,g_VP.Height/2);
ViewRender(D3DXCOLOR(1,1,0,1));
```

ChangeViewport関数は次のようになっています。

IDirect3DDevice9::SetViewport関数の実行と、それに伴うパラメーター設定しか行っていません。

```
HRESULT ChangeViewport(DWORD dwX,DWORD dwY,DWORD dwWidth,DWORD dwHeight)
{
    D3DVIEWPORT9 vp;

    vp.X=dwX;
    vp.Y=dwY;
    vp.Width=dwWidth;
    vp.Height=dwHeight;
    vp.MinZ=0;
    vp.MaxZ=1;

     if(FAILED(pDevice->SetViewport(&vp)))
     {
            MessageBox(0,"ビューポート切り替え失敗","エラー",MB_OK);
            return E_FAIL;
     }
    return S_OK;
}
```

201 異なるビューを分割した領域にレンダリングする

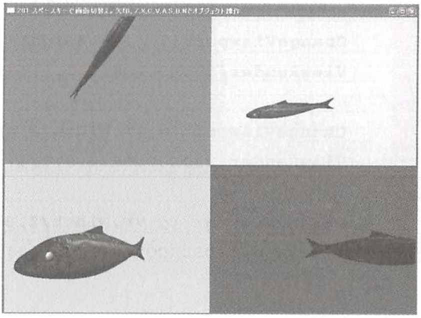
●1つのウィンドウクライアント領域を複数に分割して異なるビューでレンダリングする　初級

サンプルプロジェクト　201 画面分割その2

直前の項目の異なるビュー版です。

異なるビューにするには、それぞれの領域のレンダリングの際に、その都度、その領域用に異なるビュー・トランスフォームをかければいいのです。

▼画面分割その2



```
enum VIEW
{
    NO_SPLIT,//（通常）分割無し
    TOP,//（分割有り）上からの視点
    FRONT,//（分割有り）正面からの視点
    LEFT,//（分割有り）左からの視点
    BIRD//（分割有り）鳥瞰視点  Bird's-Eye View
};

VOID SetViewMatrix(VIEW vi)
{
    // ビュートランスフォーム（ワールド座標→カメラ座標への変換）
    D3DXMATRIX matView;
    D3DXVECTOR3 vecEyePt(0,0,0); //カメラ（視点）位置
    D3DXVECTOR3 vecLookatPt(0, 0,0);//注視位置
    D3DXVECTOR3 vecUpVec(0,1,0);//上方ベクトル定義
    switch(vi)
    {
    case NO_SPLIT:
           vecEyePt=D3DXVECTOR3(0,0,-0.3);
           break;
    case TOP:
           vecEyePt=D3DXVECTOR3(0,0.3,0.01);
           break;
    case FRONT:
           vecEyePt=D3DXVECTOR3(0,0,-0.3);
           break;
    case LEFT:
           vecEyePt=D3DXVECTOR3(-0.3,0,0);
           break;
    case BIRD:
           vecEyePt=D3DXVECTOR3(0.3,0.3,-0.3);
           break;
    }
    D3DXMatrixLookAtLH( &matView, &vecEyePt, &vecLookatPt, &vecUpVec );
    pDevice->SetTransform( D3DTS_VIEW, &matView );
}
```

```
SetViewMatrix(TOP);
ChangeViewport(0,0,g_VP.Width/2,g_VP.Height/2);
（レンダリング）

 SetViewMatrix(FRONT);
ChangeViewport(0,g_VP.Height/2,g_VP.Width/2,g_VP.Height/2);
```

```
(レンダリング)

SetViewMatrix(LEFT);
ChangeViewport(g_VP.Width/2,g_VP.Height/2,g_VP.Width/2,g_VP.Height/2);
(レンダリング)

SetViewMatrix(BIRD);
ChangeViewport(g_VP.Width/2,0,g_VP.Width/2,g_VP.Height/2);
(レンダリング)
```

202 バックミラーを表現する

●領域分割レンダリングによりバックミラーを表現する

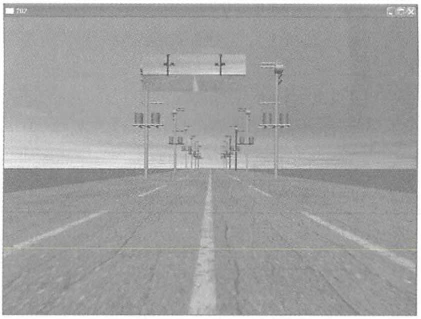
初級

サンプルプロジェクト　202 バックミラー

ビューポートにより画面分割を利用し、さらに異なるビューでレンダリングすれば“バックミラー”が簡単に実現できます。

バックミラーに見立てたビューポート矩形上に、通常のビュートランスフォームとはZ軸が正反対のビュートランスフォームをかけてレンダリングすればいいのです。Tips200とTips201と本質的に同じ処理であり、特にTips201とはやっていることは同じです。

▼実行結果



```
VOID SetViewMatrix(VIEW vi)
{
    // ビュートランスフォーム（ワールド座標→カメラ座標への変換）
    D3DXMATRIX matView;
    D3DXVECTOR3 vecEyePt(0,0,0); //カメラ（視点）位置
    D3DXVECTOR3 vecLookatPt(0, 0,0);//注視位置
    D3DXVECTOR3 vecUpVec(0,1,0);//上方ベクトル定義
    switch(vi)
    {
    case FRONT:通常のビュー
           vecEyePt=D3DXVECTOR3(0,1,z);
```

```
            vecLookatPt=D3DXVECTOR3(0,1,z+1);
            break;
        case BACK:バックミラー用のビュー
            vecEyePt=D3DXVECTOR3(0,1,z);
            vecLookatPt=D3DXVECTOR3(0,1,z-1);
            break;
        }
        D3DXMatrixLookAtLH( &matView, &vecEyePt, &vecLookatPt, &vecUpVec );
        pDevice->SetTransform( D3DTS_VIEW, &matView );

        //カメラを走らせる
        z+=0.05;
    }
```

```
    VOID Render()
    {
        //通常のレンダリング
        SetViewMatrix(FRONT);
        ChangeViewport(0,0,g_VP.Width,g_VP.Height);
        (レンダリング)
        //バックミラーのレンダリング
        SetViewMatrix(BACK);
        ChangeViewport(g_VP.Width/3,g_VP.Height/8,g_VP.Width/4,g_VP.Height/8);
        (レンダリング)

        pDevice->Present( NULL, NULL, NULL, NULL );
    }
```

```
    HRESULT ChangeViewport(DWORD dwX,DWORD dwY,DWORD dwWidth,DWORD dwHeight)
    {
        D3DVIEWPORT9 vp;

        vp.X=dwX;
        vp.Y=dwY;
        vp.Width=dwWidth;
        vp.Height=dwHeight;
        vp.MinZ=0;
        vp.MaxZ=1;

         if(FAILED(pDevice->SetViewport(&vp)))
         {
             MessageBox(0,"ビューポート切り替え失敗","エラー",MB_OK);
```

```
                return E_FAIL;
        }
    return S_OK;
}
```

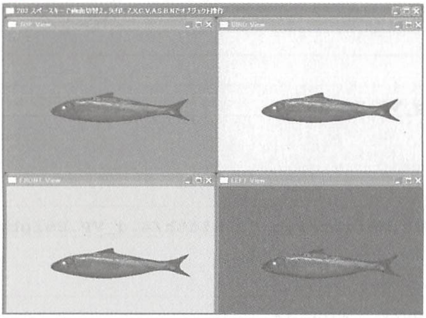
203 1つのビューを別のウィンドウにレンダリングする

複数のウィンドウ上にレンダリングする

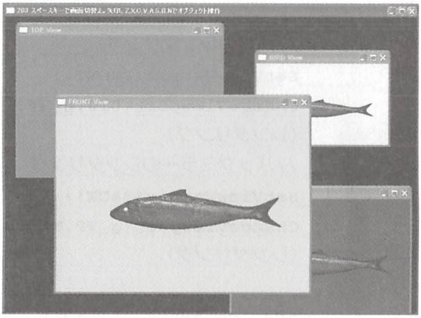
初級

サンプルプロジェクト　203 ウィンドウ分割その1

▼ウィンドウ分割その1



▼実行結果



ここでの画面分割はTips200からTips202のようにビューポート矩形で区切るのではなく、ウィンドウ自体を別々にしようというものです。つまり、画面というよりもウィンドウの分割です。

各ウィンドウは、“ウィンドウ”であるのでサイズや位置をユーザーが変更できます。

ポイントは、ビューポートが1つである代わりに、レンダーターゲットをそれぞれのウィンドウに切り替えながら、全てのウィンドウにレンダリングします。

まずウィンドウの数だけウィンドウ（ウィンドウオブジェクト）を作成しなくてはなりません。

コードは長いですが、冗長処理なので単純です。

```
// フレームウィンドウ（親ウィンドウ）の初期化
wndclass.cbSize           = sizeof (wndclass);
wndclass.style             = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc     = WndProc;
wndclass.cbClsExtra       = 0;
wndclass.cbWndExtra      = 0;
wndclass.hInstance        = hInst;
wndclass.hIcon              = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor            = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szApp;
wndclass.hIconSm            = LoadIcon (NULL, IDI_APPLICATION);
RegisterClassEx (&wndclass);
hWnd = CreateWindow (szApp,szApp,WS_OVERLAPPEDWINDOW,
               0,0,WINDOW_WIDTH,WINDOW_HEIGHT,NULL,NULL,hInst,NULL);
//子ウィンドウ（トップビューウィンドウ）の初期化
 memcpy(&wndclassChild[0],&wndclass,sizeof(WNDCLASSEX));
 wndclassChild[0].lpfnWndProc=ChildWndProc_0;
 wndclassChild[0].lpszClassName=szChild_0;
 RegisterClassEx (&wndclassChild[0]);
 hChildWnd[0] = CreateWindow (szChild_0,szChild_0,WS_CHILD |
WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN | WS_CLIPSIBLINGS |
WS_VISIBLE,0,0,WINDOW_WIDTH / 2,WINDOW_HEIGHT / 2,hWnd,NULL,hInst,NULL);
 //子ウィンドウ（フロントビューウィンドウ）の初期化
 memcpy(&wndclassChild[1],&wndclass,sizeof(WNDCLASSEX));
 wndclassChild[1].lpfnWndProc=ChildWndProc_1;
 wndclassChild[1].lpszClassName=szChild_1;
 RegisterClassEx (&wndclassChild[1]);
 hChildWnd[1] = CreateWindow (szChild_1,szChild_1,WS_CHILD |
WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN | WS_CLIPSIBLINGS |
WS_VISIBLE,0,WINDOW_HEIGHT / 2,WINDOW_WIDTH / 2,WINDOW_HEIGHT /
2,hWnd,NULL,hInst,NULL);
 //子ウィンドウ（レフトビューウィンドウ）の初期化
 memcpy(&wndclassChild[2],&wndclass,sizeof(WNDCLASSEX));
 wndclassChild[2].lpfnWndProc=ChildWndProc_2;
 wndclassChild[2].lpszClassName=szChild_2;
 RegisterClassEx (&wndclassChild[2]);
 hChildWnd[2] = CreateWindow (szChild_2,szChild_2,WS_CHILD |
WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN | WS_CLIPSIBLINGS |
WS_VISIBLE,WINDOW_WIDTH / 2,WINDOW_HEIGHT / 2,WINDOW_WIDTH / 2,WINDOW_HEIGHT
/ 2,hWnd,NULL,hInst,NULL);
 //子ウィンドウ（鳥瞰ビューウィンドウ）の初期化
memcpy(&wndclassChild[3],&wndclass,sizeof(WNDCLASSEX));
 wndclassChild[3].lpfnWndProc=ChildWndProc_3;
 wndclassChild[3].lpszClassName=szChild_3;
 RegisterClassEx (&wndclassChild[3]);
 hChildWnd[3] = CreateWindow (szChild_3,szChild_3,WS_CHILD |
WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN | WS_CLIPSIBLINGS |
WS_VISIBLE,WINDOW_WIDTH / 2,0,WINDOW_WIDTH / 2,WINDOW_HEIGHT /
2,hWnd,NULL,hInst,NULL);
```

各ウィンドウプロシージャー。これも定型コードですね。

```
//
//LRESULT CALLBACK ChildWndProc_0(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
//子ウィンドウ（トップビュー）のメッセージプロシージャー
LRESULT CALLBACK ChildWndProc_0(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
{
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
//
//LRESULT CALLBACK ChildWndProc_0(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
//子ウィンドウ（フロントビュー）のメッセージプロシージャー
LRESULT CALLBACK ChildWndProc_1(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
{
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
//
//LRESULT CALLBACK ChildWndProc_0(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
//子ウィンドウ（レフトビュー）のメッセージプロシージャー
LRESULT CALLBACK ChildWndProc_2(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
{
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
//
//LRESULT CALLBACK ChildWndProc_0(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
//子ウィンドウ（鳥瞰ビュー）のメッセージプロシージャー
LRESULT CALLBACK ChildWndProc_3(HWND hWnd,UINT iMsg,WPARAM wParam,LPARAM
lParam)
{
    return DefWindowProc (hWnd, iMsg, wParam, lParam);
}
```

スワップチェーンを用意し、追加しておきます。

```
LPDIRECT3DSWAPCHAIN9  pSwapChain[5+1]={NULL};

//追加のスワップチェーンを4つ作成　子ウィンドウ用
```

```
    for(DWORD  i=1;i<5;i++)
    {
        pDevice->CreateAdditionalSwapChain(  &d3dpp,  &pSwapChain[i]);
    }
```

レンダーターゲットを変える関数です。

```
HRESULT  ChangeRenderTarget(DWORD  dwIndex,HWND  hWnd)
{
    LPDIRECT3DSURFACE9  pBackBuffer  =  NULL;
    pSwapChain[dwIndex]->GetBackBuffer(0,D3DBACKBUFFER_TYPE_MONO,
&pBackBuffer);
    pDevice->SetRenderTarget(0,pBackBuffer  );
    pSwapChain[dwIndex]->Present(NULL,NULL,hWnd,NULL,NULL);

    return  S_OK;
}
```

ChangeRenderTarget関数をことなる引数でコールし、レンダーターゲットをそれぞれのウィンドウに一時的に切り替えて、あとは通常のようにレンダリングするだけです。

```
VOID  Render()
{
        if(boQuad)
        {
                ChangeRenderTarget(1,hChildWnd[0]);
                 (レンダリング)

                ChangeRenderTarget(2,hChildWnd[1]);
                 (レンダリング)

                ChangeRenderTarget(3,hChildWnd[2]);
              (レンダリング)

                ChangeRenderTarget(4,hChildWnd[3]);
                 (レンダリング)
        }
        else
        {
                ChangeRenderTarget(0,hWnd);
              (レンダリング)
        }
}
```

204 異なるビューを別のウィンドウにレンダリングする

●複数のウィンドウ上に異なるビューをレンダリングする

初級

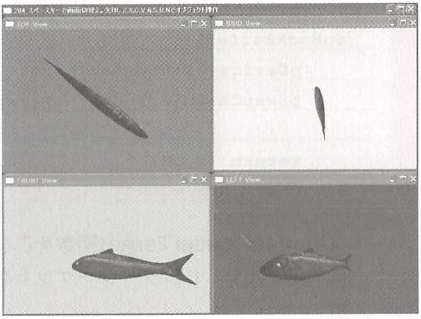
サンプルプロジェクト　204　ウィンドウ分割その2

直前の項目の異なるビュー版です。

異なるビューにするには、それぞれのウィンドウ（それぞれのレンダーターゲット）へのレンダリングの際に、その都度、そのウィンドウ用に異なるビュー・トランスフォームをかければいいのです。

直前のサンプルと異なる部分は、それぞれのレンダリングの前にSetViewMatrix関数を実行している部分だけです。

▼実行結果


204 スペースキーで画面切替え。矢印, Z,X,C,V,A,S,B,Nでオブジェクト操作
TOP View
BIRD View
FRONT View
LEFT View

```
VOID Render()
{
    if(boQuad)
    {
        SetViewMatrix(TOP);
        ChangeRenderTarget(1,hChildWnd[0]);
        RendeToCurrentWindow(D3DXCOLOR(1,0,0,1));

        SetViewMatrix(FRONT);
        ChangeRenderTarget(2,hChildWnd[1]);
        RendeToCurrentWindow(D3DXCOLOR(0,1,0,1));

        SetViewMatrix(LEFT);
        ChangeRenderTarget(3,hChildWnd[2]);
        RendeToCurrentWindow(D3DXCOLOR(0,0,1,1));

        SetViewMatrix(BIRD);
        ChangeRenderTarget(4,hChildWnd[3]);
        RendeToCurrentWindow(D3DXCOLOR(1,1,0,1));
    }
    else
    {
        SetViewMatrix(FRONT);
```

```
           ChangeRenderTarget(0,hWnd);
           RendeToCurrentWindow(D3DXCOLOR(0,1,0,1));
     }
}
```

ソース

```
enum VIEW
{
    NO_SPLIT,
    TOP,
    FRONT,
    LEFT,
    BIRD
};
```

```
VOID SetViewMatrix(VIEW vi)
{
    // ビュートランスフォーム（ワールド座標→カメラ座標への変換）
    D3DXMATRIX matView;
    D3DXVECTOR3 vecEyePt(0,0,0); //カメラ（視点）位置
    D3DXVECTOR3 vecLookatPt(0, 0,0);//注視位置
    D3DXVECTOR3 vecUpVec(0,1,0);//上方ベクトル定義
    switch(vi)
    {
    case NO_SPLIT:
          vecEyePt=D3DXVECTOR3(0,0,-0.3);
          break;
    case TOP:
          vecEyePt=D3DXVECTOR3(0,0.3,0.01);
          break;
    case FRONT:
          vecEyePt=D3DXVECTOR3(0,0,-0.3);
          break;
    case LEFT:
          vecEyePt=D3DXVECTOR3(-0.3,0,0);
          break;
    case BIRD:
          vecEyePt=D3DXVECTOR3(0.3,0.3,-0.3);
          break;
    }
    D3DXMatrixLookAtLH( &matView, &vecEyePt, &vecLookatPt, &vecUpVec );
    pDevice->SetTransform( D3DTS_VIEW, &matView );
}
```

205 フォグとは？

●Direct3Dのフォグ及び、現実のフォグを理解する

初級

fog（フォグ）は日本語で言うと（濃い）霧のことです。通常、霧というと朝霧のように水蒸気によるものを想像すると思います。フォグはそのような効果を出すエフェクトであり、固定機能シェーダーとして実装されているので、簡単に運用できます。

Direct3Dにおいて、煙霧の現実的な発生メカニズムまでを考慮しているわけではありません（3DCGソフトのプラグイン等では考慮しているものもあるかもしれません）。

そういう意味で、（あくまでDirect3Dにおける）フォグはもっと広い語彙であるヘイズ（Haze）「かすみ・もや 」と言ったほうが的を射ているかもしれません。

さて、霧の場合について言うと、空気中に浮遊する水蒸気が散乱体（scatterスキャッター）となり、光を散乱させるために霧として見えます。Direct3Dのフォグは、それを厳密にシミュレートしているわけではなく、ただ単純に“霧のような効果”を出しているだけです。実装も単純ですし、パラメーターも非常にシンプルです。結果としての見栄えだけを出力するものであって、現実のメカニズムは関係ありません。そのため逆に、“フォグ”は水蒸気のみならず、空気中の微粒子（エアロゾル）による散乱全般を表現していると言ってしまえばそうなります。水蒸気、花粉、煙粒子などがスキャッターである場合のミー散乱、分子がスキャッターである場合のレイリー散乱も、結果としての見え方だけを出力すればいいので、それらの散乱の効果をフォグで表現することができます。ただし、先に述べたとおり厳密なシミュレートではなく、単にそのように見せるだけではあります。

そうは言っても、レイリー散乱のシミュレートは別のエフェクト（Atomospher）として呼ばれることがあるので、そこまでをフォグと言うのは行き過だとは思います。

Direct3D APIとしての具体的なフォグの種類は2種類あります。「頂点フォグ」と「ピクセルフォグ」です。機能は、両者とも同じで、もちろん煙霧すなわち“もや”がかかった状態を作り出します。異なるのは、何を基準にしてフォグの濃度を決定するかということです。その名が示すとおり、頂点フォグはジオメトリの頂点を単位にフォグをかけ、ピクセルフォグはピクセルベースでフォグをかけます。

それぞれメリット・デメリットがあり、それを理解することにより状況によって使い分けることができるので、コストパフォーマンスのアップにつながります。それについては、次項206で解説しています。

Column

そんな理屈めいたことが関係あるの？

ミー散乱とレイリー散乱が登場しましたが、もちろんそんなことはDirect3Dフォグには関係がありません。あくまで現実世界での話です。ただ、現実のメカニズムを知っていることは無駄ではありません。とうよりも、必ず必要になる時がきます。混乱する可能性があるので、コラムとして書いておきます。気が向いたら読んでください。

2つの散乱は、共に光が粒子に反射・吸収される現象であり、そのメカニズムとしては同じです。異なるのは、散乱体（スキャッター）、すなわち光を反射・吸収する粒子の大きさと、それに伴う反射係数の相違です。

空が青く見えるのは、空気分子によるレイリー散乱によるものです。窒素や酸素の分子に太陽光が当たると波長の短い紫や青が散乱し、その結果青く見えるのです。他方、分子よりかなり大きなスキャッターでの霧、煙、スモッグなどはミー散乱によるものです。ミー散乱の場合は、散乱体自体がかなり大きいので全ての波長を散乱させ、結果として白く見えます。そして、もっと大きな散乱体では入射光の散乱のみならず、時には粒子内部での拡散反射光さえも目に届き、粒子自体の色が見えるときがあります（例：黄砂）。

拡散反射とは、本質的には光が物体内部（の散乱体）で均等に散乱する現象です。（拡散と散乱は異なる概念です）。ちなみに、よく「表面がザラザラしている物体では拡散反射が起こり易い」という表現は、正解でもあり間違いでもある可能性があり、何れにしても正確ではありません。その理由までここでは書きません。

なお、ミー散乱が起こっている場所でも空気があるわけですからレイリー散乱も同時に起こっています。しかし、ミー散乱のほうがレイリー散乱より相対的に非常に強いのでミー散乱の効果しか視認できません。つまり微粒子の多い地上付近ではミー散乱が支配的なだけであって、レイリー散乱も起こっているということです。逆に上空はチリが少なく澄んでいますからミー散乱に邪魔されることなく空気構成分子によるレイリー散乱を確認することができるわけです。

さて、“フォグ”にとってそんなことはどうでもいいことのように思えますが、厳密な計算式ではないものの、それに近似させたシミュレートを行っているゲームがあります。鬼武者3（開発：カプコン）は、単なるフォグではなく、これらの散乱をモデルにしたPlayStation2上でのシェーディングモデルをSIGGRAPH2003で発表しています。やはり現実のメカニズムを把握しておくことは有益であることは間違いないでしょう。今は不可能でも、ハードが進化するにつれ、リアルタイムで厳密な計算が可能な環境になる可能性のほうが高いと思いますし、実際3DCGの様々な手法は、現実の“ラフなモデル”からハードの向上による“現実により近似したモデル”へのレベルアップという歴史を辿ってきているのは紛れも無い事実ですから。

206 頂点フォグの問題点

●頂点ベースのフォグにおける不具合を把握する

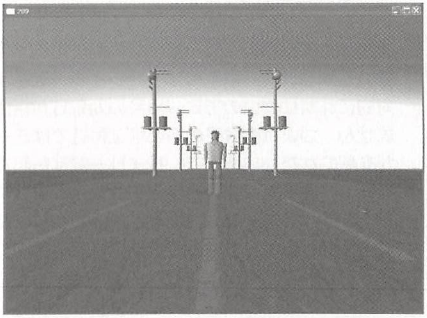
初級

サンプルプロジェクト　206　頂点フォグの不具合

頂点フォグは頂点を基準にするので、ジオメトリの状態によってはしばしば不具合が発生します。ジオメトリの状態とは、頂点数が少なく、かつ、サイズが大きなジオメトリです。このサンプルでは頂点フォグによる不具合が起きるような設定を意図的に行っています。

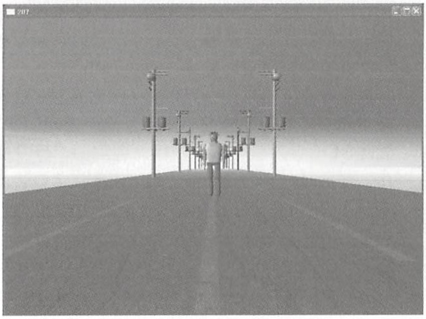
まずは、ピクセルフォグによる正常なレンダリングを見てください。

▼実行結果（正常なレンダリング）



次に、頂点フォグによる不具合が出ているレンダリングを見てください。

▼実行結果（頂点フォグによる不具合）



どこが不具合かというと、草原（道路わき）を見てください。全てにフォグがかかってしまい、全体が白くなっています。それにより道路だけが宙に浮いているように見えます。道路自体も本来フォグが適用されない手前部分にフォグがかかってしまっています。

実は草原の部分はたった1枚のポリゴンしかありません。草原はテクスチャを貼れば1枚のポリゴンでも事足りるのでそうしました。しかし、これが頂点フォグの弱点を露呈してしまっています。

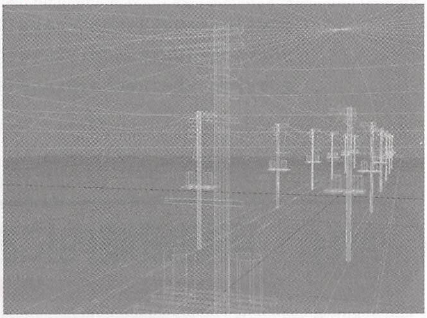
草原部分は1枚のポリゴンですから、最も手前にある表面でも、4つの頂点のうち2つははるか彼方にあるのです。草原部分は、領域としては最も大きな部分にも関らず、頂点フォグは頂点に対してフォグをかけるので、遥か彼方の頂点でも、その頂点を使用しているポリゴンに適用されてしまい、表面がフォグの範囲外にあろうとフォグがかかってしまいます。

次図はLightwaveにおけるシーンですが、各部分でのポリゴン数が大まかに分かると思います。

▼実行結果（Lightwaveにおけるシーン）

電柱部分はかなりのポリゴン数です。空（上部をドーム状に覆っている部分）もけっこうなポリゴン数です。道路はポリゴン数が少ないですが、運よくサンプルの設定ではさほど不具合は露呈していません。でも、やはりフォグの範囲(Start140〜End200)の外であってもうっすらフォグがかかってしまっています。

▼実行結果



207 開始点と終了点によりフォグをかける

フォグの範囲を指定する（距離ベースフォグ）

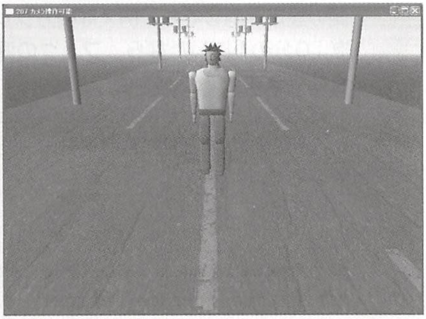
初級

サンプルプロジェクト　207　範囲指定フォグ

フォグの開始点（Start）と終了点（End）を指定することにより、フォグの範囲を特定することができます。

コードは次のようになります。

▼実行結果



```
//フォグの設定
{
    FLOAT StartPos=20;
    FLOAT EndPos=100;

    pDevice->SetRenderState(D3DRS_FOGENABLE, TRUE);
    pDevice->SetRenderState(D3DRS_FOGCOLOR,
D3DCOLOR_ARGB(255,255,255,255));
```

```
        pDevice->SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_NONE );
        pDevice->SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_LINEAR );
        pDevice->SetRenderState(D3DRS_FOGSTART, *(DWORD *)(&StartPos));
        pDevice->SetRenderState(D3DRS_FOGEND,    *(DWORD *)(&EndPos));
    }
```

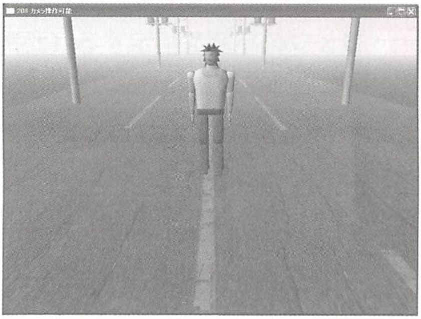
208 可視領域内における密度によりフォグをかける

●ニアクリップとファークリップ間でのフォグ密度を指定する

初級

サンプルプロジェクト　208　密度ベースフォグ

▼実行結果



開始点と終了点を絶対値で明示的に指定するのではなく、ある範囲の密度（Density）を指定してフォグをかけることもできます。

コードは次のようになります。なお、何を基準にした密度なのかと言うと、ビュー視錐台の前方クリップ面と後方クリップ面の距離を基準にした密度です。したがって、フォグ側の設定が同じでも、例えば後方クリップ面を10分の1にしたなら、フォグの濃さは強くなります。

前方クリップ面と後方クリップ面とは、射影行列を作成する際にパラメーターとして渡す値です。

例えば、

```
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, (FLOAT)WINDOW_
WIDTH/(FLOAT)WINDOW_HEIGHT, 0.2f, 1000.0f );
```

では第4引数の0.2fが視点から前方クリップ面（ニアクリップ）までの距離、第5引数の1000.0fが視点から後方クリップ面（ファークリップ）までの距離を意味しています。ファークリップからニアクリップを引いた距離が、視錐台の長さになります。通常はニアクリップを1以下に設定するので、ほぼファークリップと同じ値になるでしょう。

```
//フォグの設定
 {
     FLOAT       Density=0.03f;
```

```
    pDevice->SetRenderState(D3DRS_FOGENABLE, TRUE);
    pDevice->SetRenderState(D3DRS_FOGCOLOR, D3DCOLOR_ARGB(255,255,255,255));
    pDevice->SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_NONE );
    pDevice->SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_EXP );
    pDevice->SetRenderState(D3DRS_FOGDENSITY, *(DWORD *)(&Density));
}
```

209 頂点フォグ

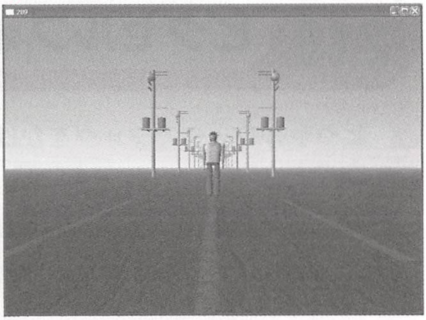
頂点フォグでフォグをかける

初級

サンプルプロジェクト　209　頂点フォグ

頂点フォグをかけるには、次のようなコードになります。

▼実行結果



```
//フォグの設定 指数的とは距離に対して。距離とはプロジェクショントランスフォームの最大クリップ距離に依存する
{
    FLOAT       Density=0.01f;
    DWORD Color=D3DCOLOR_ARGB(255,255,255,255);

    pDevice->SetRenderState(D3DRS_FOGENABLE, TRUE);
    pDevice->SetRenderState(D3DRS_FOGCOLOR, Color);
    pDevice->SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_NONE );
    pDevice->SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_EXP );
    pDevice->SetRenderState(D3DRS_FOGDENSITY, *(DWORD *)(&Density));
}
```

なお、そのグラフィック環境で頂点フォグをサポートしているかどうかを事前に調べたい場合は次のようなコードになります（頂点フォグであれば、ほとんど全てのグラフィックカードでサポートしているとは思いますが）。

```
//フォグが使えるか
D3DCAPS9 caps;
ZeroMemory(&caps,sizeof(D3DCAPS9));
pDevice->GetDeviceCaps(&caps);
if( ( caps.RasterCaps & D3DPRASTERCAPS_FOGVERTEX ) == 0 )
{
     if(MessageBox(0,"頂点フォグが使えない可能性があります。続行しますか?",
               "",MB_ABORTRETRYIGNORE) == IDABORT)
     {
               return E_FAIL;
     }
}
```

210 ピクセルフォグ

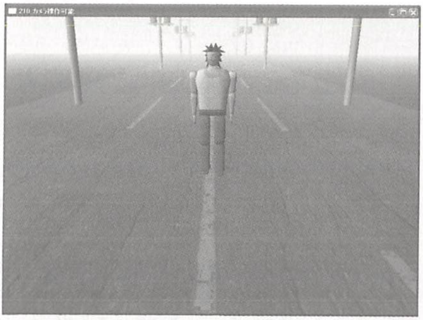
●ピクセルフォグでフォグをかける

初級

サンプルプロジェクト　210　ピクセルフォグ

ピクセルフォグをかけるには、次のようなコードになります。

▼実行結果



```
//フォグの設定
{
     FLOAT          Density=0.03f;
```

```
    pDevice->SetRenderState(D3DRS_FOGENABLE, TRUE);
    pDevice->SetRenderState(D3DRS_FOGCOLOR,
D3DCOLOR_ARGB(255,255,255,255));
    pDevice->SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_NONE );
    pDevice->SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_EXP );
    pDevice->SetRenderState(D3DRS_FOGDENSITY, *(DWORD *)(&Density));
  }
```

なお、そのグラフィック環境でピクセルフォグをサポートしているかどうかを事前に調べたい場合は、次のようなコードになります。

```
//フォグが使えるか
D3DCAPS9 caps;
 ZeroMemory(&caps,sizeof(D3DCAPS9));
 pDevice->GetDeviceCaps(&caps);
 if( ( caps.RasterCaps & D3DPRASTERCAPS_FOGTABLE ) == 0 )
 {
     if(MessageBox(0,"ピクセルフォグが使えない可能性があります。続行しますか?",
           "",MB_ABORTRETRYIGNORE) == IDABORT)
     {
           return E_FAIL;
     }
 }
```

ぼかし

211 フルシーンアンチエイリアシング(FSAA)のサポートの有無を調べる

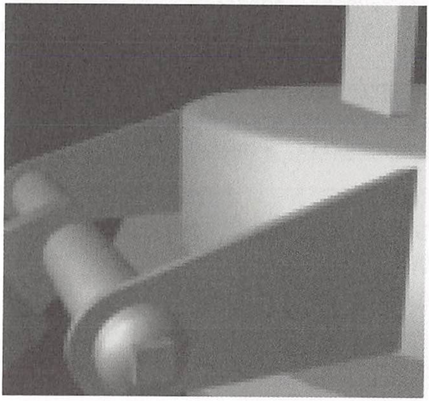
●ハードウェアのマルチサンプリング能力を調べる　初級

FSAAはフルシーンアンチエイリアシング(Full Scene Anti Aliasing)の略称です。その名のとおりシーン全体つまり画面単位でアンチエイリアスをかけることです。アンチエイリアスとは、最終的なピクセル境界のジャギーをぼかして目立たなくさせることです。

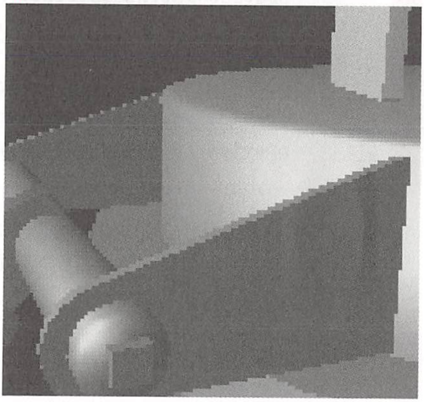
DirectX6の時代からDirect3DにはFSAAが固定機能シェーダーレベルで実装されています（そもそもDirectX6の時代はまだプログラマブルシェーダーが無かったので）。

ハードウェアがFSAAをサポートしているかどうかを判断するコードは次のようになります。なお、「FSAA」は「マルチサンプル」によって実現しますので、FSAA能力の有無はマルチサンプルが出来るかどうかということになります。

▼アンチエイリアス有り



▼アンチエイリアス無し



```
D3DDISPLAYMODE dm;
 pD3d->GetAdapterDisplayMode(0,&dm);
 if( FAILED(pD3d->CheckDeviceMultiSampleType( D3DADAPTER_DEFAULT,
     D3DDEVTYPE_HAL , dm.Format, FALSE,D3DMULTISAMPLE_2_SAMPLES, NULL ) ) )
 {
     MessageBox(0,"マルチサンプル未対応","",MB_OK);
     return E_FAIL;
 }
```

212 フルシーンアンチエイリアシング(FSAA)をかける

●SetRenderStateのD3DRS_MULTISAMPLEANTIALIASをTRUEにしてマルチサンプルを有効にする　初級

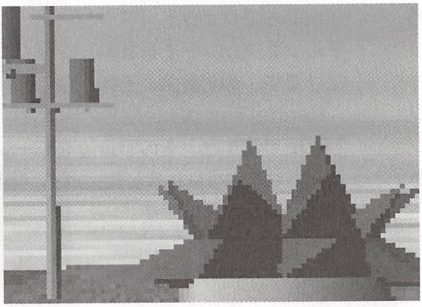
FSAAの適用は、たった1行です。

```
pDevice->SetRenderState(D3DRS_MULTISAMPLEANTIALIAS,TRUE);
```

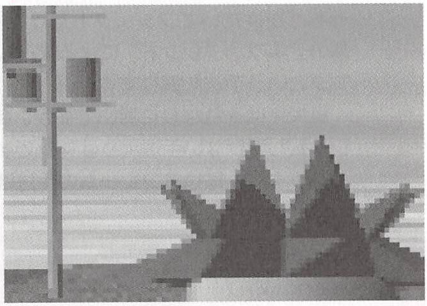
FSAAの効果を確認するために、このステートを無効（FALSE）にした場合のレンダリングと比較してみましょう。

```
pDevice->SetRenderState(D3DRS_MULTISAMPLEANTIALIAS,FALSE);
```

▼実行結果（FALSE）



▼実行結果（TRUE）



One Point

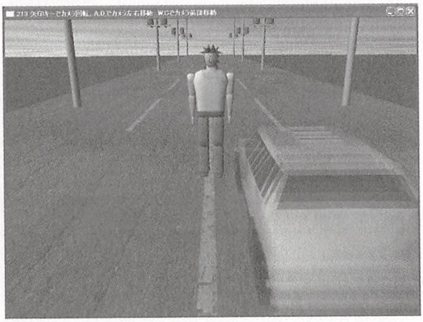
FSAAをサポートしているかどうかを確認する方法については、Tips211を参照してください。

213 実用的モーションブラー

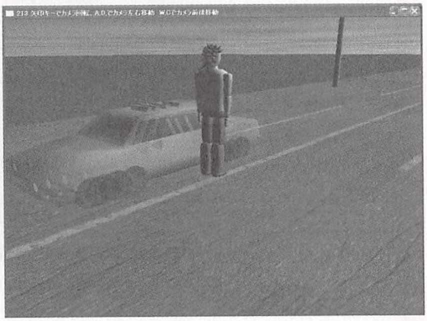
●モーションブラーをアルファブレンドによりシンプルに実装する 上級

サンプルプロジェクト 213 モーションブラー

▼実行結果


213 矢印キーでカメラ回転、A,D,でカメラ左右移動 W,Cでカメラ前後移動

▼実行結果


213 矢印キーでカメラ回転、A,D,でカメラ左右移動 W,Cでカメラ前後移動

通常、モーションブラーを実装すると"極度に"遅い実装となってしまいます。

"シーン全体"を"何枚も"重ね、しかもその1枚1枚がピクセルシェーダーによる重たい処理であるのがその原因です。結果としての全体速度は悲惨なものになり、とても実

用的なものとは言えません。ここでは、ブラーを“かけたいオブジェクトだけ”を、“通常のアルファブレンド”による高速レンダリングという手法を採ります。

見栄えはそれなりに悪いですが、毎秒5フレームも稼げない手法よりはマシです（マシというより、毎秒数フレームの手法は使いものになりません）。

仕組みは単純です。1フレーム内でブラー対象のオブジェクト（タクシーメッシュ）をアルファブレンドで複数回レンダリングするだけです。透明度は、その1.0/回数です。

RenderThingBlur関数のBOOL　BlurをTRUEで渡すと、そのThingをモーションブラーさせるようにしました。ポイント部分は、関数内の次の部分です。

```
//ブラーあり
 if(Blur)
 {
     pDevice->SetRenderState ( D3DRS_ALPHABLENDENABLE, TRUE );

     pDevice->SetRenderState ( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
     pDevice->SetRenderState ( D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA );
     pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1,D3DTA_DIFFUSE);
     pDevice->SetTextureStageState(0,D3DTSS_ALPHAARG2,D3DTA_TEXTURE);

     //見る方向によって前回位置と現在位置の関係を逆転させる（アルファブレンドがおかしくならないように）
     D3DXVECTOR3 Tmp;
     D3DXVECTOR3 PosTmp=Pos;
     D3DXVECTOR3 PrevPosTmp=PrevPos;
     D3DXVec3TransformCoord(&PosTmp,&PosTmp,&matView);
     D3DXVec3TransformCoord(&PrevPosTmp,&PrevPosTmp,&matView);
     D3DXVECTOR3 vObjDir=PosTmp - PrevPosTmp;
     if(vObjDir.z >=0)//前方方向に向かっている（遠ざかっている）
     {
          Tmp=Pos;
          Pos=PrevPos;
          PrevPos=Tmp;
     }

     for(k=0;k<BLUR_DEPTH;k++)
     {
          //ワールドトランスフォーム（絶対座標変換）
          D3DXMATRIX matWorld,matPosition;
          D3DXMatrixIdentity(&matWorld);
D3DXMatrixTranslation(&matPosition,PrevPos.x,PrevPos.y,PrevPos.z);
          D3DXMatrixMultiply(&matWorld,&matWorld,&matPosition);
```

```
        pDevice->SetTransform( D3DTS_WORLD, &matWorld );

        for(  i=0; i<pThing->dwNumMaterials; i++ )

        {
            D3DMATERIAL9 mtrl;
            memcpy(&mtrl,&pThing->pMeshMaterials[i],sizeof(D3DMATERIAL9));
            mtrl.Diffuse.a= (FLOAT)1 / (FLOAT) BLUR_DEPTH +0.1;

            pDevice->SetMaterial( &mtrl );
            pDevice->SetTexture( 0,pThing->pMeshTextures[i] );

            pThing->pMesh->DrawSubset( i );

        }
        PrevPos+=(Pos-PrevPos) / (FLOAT)BLUR_DEPTH;
    }
}
```

この手法で気をつけなければならないのは、アルファブレンドが特定の方向から見た場合、おかしなことになってしまうのを防ぐことです。

こちらに向かっている場合とこちらから遠ざかっている場合に、1フレーム内の複数レンダリングを開始する場所を逆転させないとアルファブレンディングゆえの不具合が起こります。遠ざかっているときは大丈夫でも、近づいているときに背景まで透けてしまいます。

One Point

アルファブレンドの本質的な問題については、Tips179「カメラ位置に依存しないアルファブレンディングを実現する」を参考にしてください。

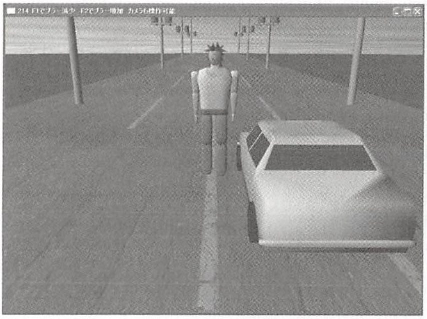
214 シーン全体をぼかす

●ポストプロセスによりシーン全体にぼかしを加える

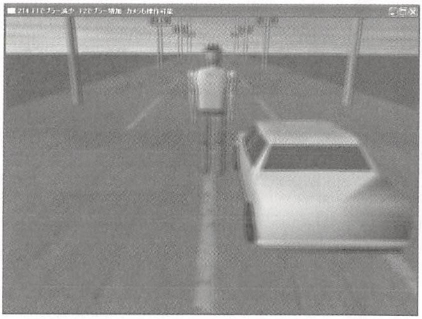
上級

サンプルプロジェクト 214 全体ぼかし

▼ぼかし無し



▼ぼかし強め



事後的に何らかの処理（主にエフェクト）を加えることを「ポスト・プロセッシング(post processing)」と言い、その処理を「ポストプロセス（post process)」と言います。3DCGに限らずサウンド業界でも、全ての処理が終わった後にかける付加的エフェクトをそう呼びます。

3DCGでのポストプロセスによるエフェクト（以降ポストエフェクトと表記）は具体的には、シーンの3Dレンダリングが終わった後に2D的なピクセル操作をして様々なエフェクト効果を付け加えることが主なものです。

ここでは、シーンをテクスチャ上にレンダリングし、そのテクスチャを2D操作でぼかすことを行います。2D操作ですから3Dレンダリング過程では出来ません。一旦シーンを全てテクスチャ上にレンダリングします。テクスチャ上にレンダリングするためには、レンダーターゲットをテクスチャになくてはなりません。また、そのテクスチャを表示するためには、再度レンダーターゲットを元のバックバッファに戻す操作が必要です。

バックバッファと深度バッファ（Zバッファ）のポインタを保存しておきます。

```
pDevice->GetRenderTarget(0,&pBackBuffer);
pDevice->GetDepthStencilSurface(&pZBuffer);
```

レンダーターゲットをテクスチャに切り替えます。

```
LPDIRECT3DSURFACE9 pSurf = NULL;
 pTargetTexture->GetSurfaceLevel(0,&pSurf);
 pDevice->SetRenderTarget(0,pSurf );
```

```
pDevice->SetDepthStencilSurface(pTextureZBuffer);
SAFE_RELEASE(pSurf);
```

ここでシーンを通常のようにレンダリングします。シーンはテクスチャ上にレンダリングされます。

ここでプログラマブルシェーダーによりテクスチャに2D的処理を加えます（ばかしを加えます）。

```
///////////////////////
//型定義
///////////////////////
struct VS_OUT
{
    float4 Pos : POSITION;
    float2 TC0 : TEXCOORD0;
    float2 TC1 : TEXCOORD1;
    float2 TC2 : TEXCOORD2;
    float2 TC3 : TEXCOORD3;
};

///////////////////////
//グローバル
///////////////////////
float TexWidth=1024;
float TexHeight=1024;
float Level=2;
texture BlurTex;

sampler BlurSampler = sampler_state
{
    Texture = (BlurTex);
    MinFilter = LINEAR;
     MagFilter = LINEAR;
    MipFilter = NONE;
    AddressU = Clamp;
    AddressV = Clamp;
};

///////////////////////
//バーテックス・シェーダー
///////////////////////
VS_OUT VS(float4 Position : POSITION, float2 TC : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Pos = Position;
    float DeltaH=Level/TexWidth;
    float DeltaV=Level/TexHeight;
    Out.TC0 = TC + float2(-DeltaH, -DeltaV);
    Out.TC1 = TC + float2(DeltaH, -DeltaV);
    Out.TC2 = TC + float2(-DeltaH,DeltaV);
    Out.TC3 = TC + float2(DeltaH,DeltaV);

    return Out;
}

///////////////////////
//ピクセル・シェーダー
///////////////////////
float4 PS( VS_OUT In) : COLOR
{
    float4 Color0=tex2D(BlurSampler,In.TC0);
    float4 Color1=tex2D(BlurSampler,In.TC1);
    float4 Color2=tex2D(BlurSampler,In.TC2);
    float4 Color3=tex2D(BlurSampler,In.TC3);

    float4 FinalColor=(Color0+Color1+Color2+Color3)/4.0f;
    return FinalColor;
}

///////////////////////
// テクニック
///////////////////////
technique tecBlur
{
    pass p0
    {
    VertexShader = compile vs_2_0 VS();
    PixelShader  = compile ps_2_0 PS();
    }
}
```

シェーダー内容の詳細については割愛し、仕組みだけ述べます。

仕組みは、ピクセルシェーダーがピクセル色を決定するさいに拾うテクセル数を増やす（テクセル範囲を広げる）というものです。それにより、“ぼけ”が生まれれます。コードが多く複雑に見えますが、原理はそれに尽きます。

なお、簡単のため、および実行速度の向上のため、いささか乱暴なぼかしを行っています。テクセルを拾う範囲を広げすぎると、物体の分身ができたようなぶれ方をします。そのような見た目の悪さは割り切って考えてください。

これでほとんど完了です。せっかく出来上がったテクスチャも画面に映らなければ意味

がないので、ぼけたシーンのテクスチャを画面に出力するのが以降の処理です。

レンダーターゲットを元に戻します。

```
pDevice->SetRenderTarget(0,pBackBuffer  );
pDevice->SetDepthStencilSurface(pZBuffer);
SAFE_RELEASE(pBackBuffer);
SAFE_RELEASE(pZBuffer);
```

ここでテクスチャを板ポリゴンに貼って、そのポリゴンをレンダリングします。シーンの絵が貼ってある板ポリゴンをちょうど画面いっぱいに重なるようにすれば、あたかもシーンが最初からバックバッファにレンダリングされたのと同じように見えるというわけです。

215

距離によりぼかし具合を変化させる

●テクスチャにZ深度を記録し、それを“ぼかし”度合いに利用する

上級

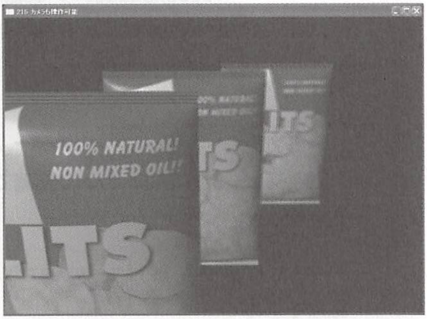
サンプルプロジェクト 215 深度テクスチャ

Tips214では、シーンのありとあらゆるものを全てぼかしました。ここでは、各オブジェクトのカメラからの距離によってぼかし具合を変化させます。

これを実現するための方法として、テクスチャに各オブジェクト（正確に言えば各頂点）に対するカメラからのZ深度を記録し、ぼかし工程でそのテクスチャを利用するというものです。

シェーダーのコードは次のとおりです。

▼実行結果



```
////////////////////////////////////////////
//深度テクスチャ作成用 バーテックス・シェーダー
////////////////////////////////////////////
VS_OUT VS_Depth(float4 Position : POSITION, float2 TC : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Pos=mul(Position,matWVP);
    float4 Pos=mul(Position,matWV);

    float Length=length(Pos);

    Out.Col.r=Length / FarClip;

    return Out;
}

/////////////////////////////////////
//深度テクスチャ作成用 ピクセル・シェーダー
/////////////////////////////////////
float4 PS_Depth( VS_OUT In) : COLOR
{
    return In.Col;
}

/////////////////////////////////////
// 深度テクスチャ作成用 テクニック
/////////////////////////////////////
technique tecMakeDepthImage
{
    pass p0
    {
         VertexShader = compile vs_2_0 VS_Depth();
       PixelShader  = compile ps_2_0 PS_Depth();

    }
}
```

頂点シェーダーでは、カメラ空間での座標を求め、

```
float4 Pos=mul(Position,matWV);
```

距離を求めて、

```
float Length=length(Pos);
```

その距離を0.0から1.0にするために、最大クリップ面までの距離で割っています。言い換えると、Ｚの最大値で割ることにより正規化しています（正確にするには、最小クリップ面までの距離を引いてから最大クリップ面距離で割ります）。

```
Out.Col.r=Length / FarClip;
```

出力カラーのうちr成分に記録します。したがって、この後でこのテクスチャにより深度を取り出すには、r成分を調べることになります（rではなくgでもbでもaでも良かったのですが、適当にrにしました）。

ピクセルシェーダーでは、なにもしてません。頂点シェーダーからもらったカラーをスルーパスしてるだけです。つまり頂点シェーダーで全ての仕事をしているということです。

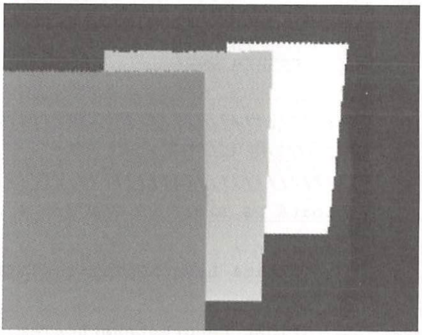
return In.Col;

ぼかし用の頂点シェーダー、ピクセルシェーダーは直前の項目と同一です。いちおう掲載しておきます。

```
/////////////////////////////////////////
//ぼかし用 バーテックス・シェーダー
/////////////////////////////////////////
VS_OUT VS_Blur(float4 Position : POSITION, float2 TC : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Pos = Position;
    Out.TC=TC;


    return Out;
}
/////////////////////////////////////////
//ぼかし用 ピクセル・シェーダー
/////////////////////////////////////////
float4 PS_Blur( VS_OUT In) : COLOR
{
    float4 LengthOnTex=tex2D(DepthSampler,In.TC);

    float Length=LengthOnTex.r;

    Length*=Level;

    float2 TC0,TC1,TC2,TC3;
    float DeltaH=Length/TexWidth;
    float DeltaV=Length/TexHeight;
    TC0=In.TC+float2(-DeltaH,-DeltaV);
    TC1=In.TC+float2(DeltaH,-DeltaV);
    TC2=In.TC+float2(-DeltaH,DeltaV);
    TC3=In.TC+float2(DeltaH,DeltaV);

    float4 Color0=tex2D(SceneSampler,TC0);
    float4 Color1=tex2D(SceneSampler,TC1);
    float4 Color2=tex2D(SceneSampler,TC2);
    float4 Color3=tex2D(SceneSampler,TC3);
    return (Color0+Color1+Color2+Color3)/4.0f;
}
//////////////////////////////////////////
// ぼかし用 テクニック
//////////////////////////////////////////
technique tecBlur
{
    pass p0
    {
          VertexShader = compile vs_2_0 VS_Blur();
        PixelShader  = compile ps_2_0 PS_Blur();

    }
}
```

深度テクスチャを可視化したイメージは次のようになっています。距離によって値が異なるのが確認できるでしょう。奥に行くほど白く（値が大きく）なっています。

▼実行結果



216　被写界深度（DOF）効果を出す

焦点距離、開放値を反映させてDOF効果を出す

上級

サンプルプロジェクト　216 DOF

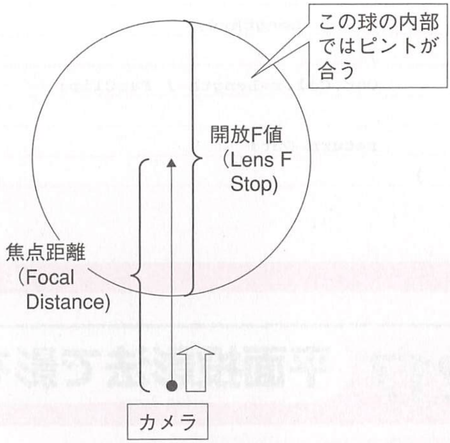
直前のサンプルでも、ある種のDOF（被写界深度）効果と言ってもいいのですが、やはりDOF効果であるためには焦点距離と開放F値を設定できなければならないので、直前のサンプルをちょっとだけ改造しました。主要な原理はまったく同じです。

直前のサンプルと異なるのは、「深度テクスチャ作成」部分だけです。その後の処理は

まったく同じです。

焦点距離（Focal Distance）とはピントの合う範囲の中心までの距離です。開放F値（Lens F Stop）とは焦点距離の前後どこまでをピントの合う範囲にするかの値です。図にすると右のようになります。

深度テクスチャ作成用の頂点シェーダーが異なります。



```
float FocalDistance=1;//焦点距離
float LensFStop=1;//開放F値
float FarClip=20;//最大クリップ面までの距離
```

```
//////////////////////////////////////////
//深度テクスチャ作成用 バーテックス・シェーダー
//////////////////////////////////////////
VS_OUT VS_Depth(float4 Position : POSITION, float2 TC : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos=mul(Position,matWVP);
    float4 Pos=mul(Position,matWV);
```

```
    float Length=length(Pos);

    if(Length<FocalDistance-LensFStop/2)//焦点領域より前方
    {
        Length=abs(FocalDistance-Length);
    }
    else if(Length>FocalDistance+LensFStop/2)//焦点領域より後方
    {
        Length=abs(Length-FocalDistance);
    }
    else//焦点領域内（ブレなし）
    {
            Length=0;
        }
        Out.Col.r=Length / FarClip;

        return Out;
}
```

影

217 平面投影法で影を作る

平面投影で元ジオメトリを“ぺちゃんこ”にして影を作る

初級

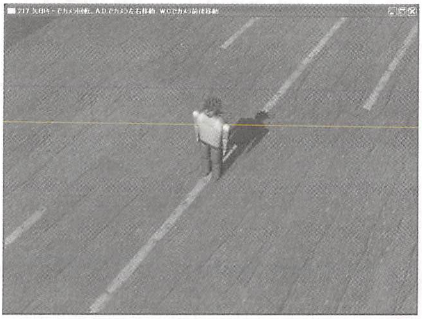
サンプルプロジェクト 217 平面投影シャドウ

変面投影シャドウは影を付ける技法で簡単な方法の一つです。最も簡単な手法はいわゆる「丸影」と呼ばれるもので、ジオメトリの形状に関らず固定形状（丸い影）を影とするものですが、それではあまりにも見栄えが悪いので、本書では平面投影シャドウから入ります。

平面投影シャドウの原理は、影を落とすジオメトリをまず普通にレンダリングし、その後で特殊な行列を掛けてジオメトリを平面つまり“ぺしゃんこ”にします。そのぺしゃんこにしたジオメトリのマテリアルを黒っぽくして、さらにジオメトリの足元付近に適当に配置して影として見せます。このような安いトリックなのでジオメトリ自体に影は落ちません（セルフシャドウは発生しません）。

なお、ジオメトリを2つ用意するのではなく、あくまでもジオメトリは一つだけです。影をレンダリングするときのワールド変換で一時的にジオメトリを平べったくするだけです。

▼実行結果


217 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Cでカメラ前後移動

```
RenderThing(&Thing[1]);
RenderShadow(&Thing[1]);
```

RenderThing関数は通常のレンダリング関数です。その直後にRenderShadow関数により影をレンダリングします。RenderShadow関数の中身は次のとおりです。

```
VOID RenderShadow(THING* pThing)
{
    //影用の射影マトリクスを作成
    D3DXMATRIX mShadow;
    D3DXVECTOR3 vPoint(0,-0.3,0);
    D3DXVECTOR3 vNormal(0,-1,0);
    D3DXPLANE GroundPlane;
    D3DXPlaneFromPointNormal(&GroundPlane,&vPoint,&vNormal);
    D3DXMatrixShadow(&mShadow,&vLightDirection,&GroundPlane);

    mShadow=pThing->matWorld*mShadow;
    //影用マテリアルの設定
    D3DMATERIAL9 Material;
    ZeroMemory( &Material, sizeof(D3DMATERIAL9));
    Material.Emissive.a = 1.0f;
    Material.Emissive.r = 0.1f;
    Material.Emissive.g = 0.1f;
    Material.Emissive.b = 0.1f;
    //影としてレンダリング
    pDevice->SetTransform( D3DTS_WORLD, &mShadow );
    pDevice->SetMaterial( &Material );
    for( DWORD i=0; i<pThing->dwNumMaterials; i++ )
     {
          pThing->pMesh->DrawSubset( i );
     }
}
```

まずジオメトリを“つぶす・平べったくする”行列、mShadowを作成します。それをジオメトリのその時点でのワールド行列に掛け、それをレンダリングパイプラインに登録します。これでレンダリングすればジオメトリは平ぺったくレンダリングされます。

そして、影ですから色が付いていては困るので、影として暗めのマテリアルも設定します。

One Point

影にアルファブレンドをかけたくなるでしょうが、影を半透明にするとポリゴンの重なり具合が見えてしまい、平面投影した影ということがばれてしまいます。平面投影シャドウの場合は割り切って、影が不透明でも我慢しましょう。

218　シャドウボリューム法で影を付ける

●ジオメトリの離隔となる頂点のみを引き伸ばした領域（シャドウボリューム）を作成して影を作る

上級

サンプルプロジェクト　218　シャドウボリューム

シャドウボリューム法はTips222シャドウマップ法と並び、本格的なシャドウを生成できます。本格的というのは、他のジオメトリに影が落ちるのはもちろん、加えてセルフシャドウ（自分の影が自分自身にも落ちる）もかかります。そして、影は不透明ではなく、影が落ちている部分のテクスチャも見ることができます。つまり、平面投影法の欠点を全て解消した完全な影を生成する技法です。

仕組みは、次のとおりです。

①まずジオメトリの輪郭部分の頂点を抽出し、その頂点をライト方向に引き伸ばしたような領域を作成します。これをシャドウボリュームと呼びます。

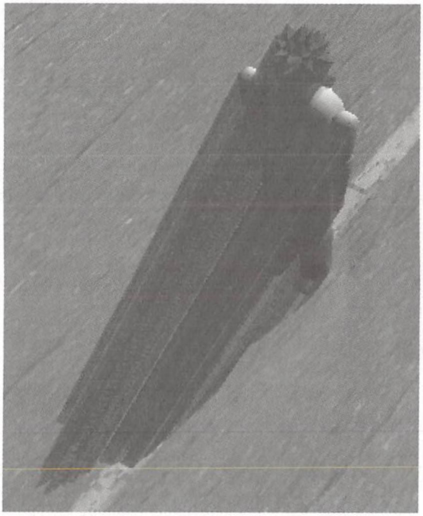
②ステンシルバッファを利用して、ボリュームとシーン内全てのポリゴン（もちろん影を落とすジオメトリも含む）での重なりを調べて、重なっている部分を影と判断します（上手いことを考えたものです）。

③画面全体を覆う（かつ最全面に）板ポリゴンを置き、板ポリゴン（のテクスチャ）上にシーンをレンダリングする。こうすることにより、②の判定結果で影と判断した部分を影として簡単にレンダリングできます。つまり、シャドウボリューム法は最終的にな仕組みとしてはポストプロセスです。

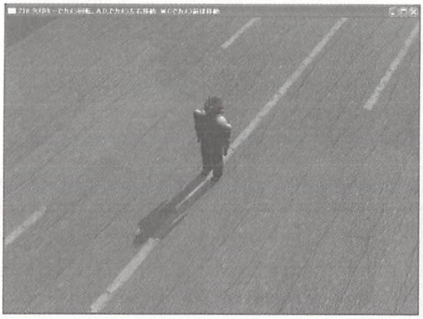
サンプルで、これらの手順に関るコード部分は次のとおりです（基本的にシャドウボリュームに関る部分はShadowVolume.cppファイルにまとめています。

言い換えると、ShadowVolume.cpp内のコードはすべてポイント部分です、また、Main.cpp側のコードは本質ではありません）。

▼シャドウボリューム



▼実行結果


218 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Sでカメラ前後移動

●①の処理をしている部分

```
//グローバル
DWORD g_dwVerticesAmt=0;
D3DXVECTOR3 g_pVertices[60000];
```

●ボリュームを作成する関数

```
HRESULT CreateVolume(LPD3DXMESH pMesh, D3DXVECTOR3 vLightDir)
{
    g_dwVerticesAmt=0;

    D3DXVECTOR3 vVertex0;
    D3DXVECTOR3 vVertex1;
    D3DXVECTOR3 vVertex2;
    D3DXVECTOR3 vVertex3;
    D3DXVECTOR3 vVertex4;
    MESH_VERTEX *pVertices;
    WORD        *pIndices;

    DWORD dwPolyAmt = pMesh->GetNumFaces();
    WORD *pEdges = new WORD[dwPolyAmt*4];

    pMesh->LockVertexBuffer( 0L, (LPVOID*)&pVertices );
    pMesh->LockIndexBuffer( 0L, (LPVOID*)&pIndices );

    DWORD dwEdgesAmt=0;
    // 全ポリゴンについてライトベクトルとの角度を調べて、輪郭となるポリゴンを抽出
    for( DWORD i = 0; i < dwPolyAmt; i++ )
    {
        WORD wPolyIndex0 = pIndices[3*i+0];
        WORD wPolyIndex1 = pIndices[3*i+1];
        WORD wPolyIndex2 = pIndices[3*i+2];
    vVertex0 = pVertices[wPolyIndex0].Pos;
    vVertex1 = pVertices[wPolyIndex1].Pos;
    vVertex2 = pVertices[wPolyIndex2].Pos;

    D3DXVECTOR3 vNormal;
        D3DXVECTOR3 vCross1(vVertex2-vVertex1);
        D3DXVECTOR3 vCross2(vVertex1-vVertex0);
        D3DXVec3Cross( &vNormal, &vCross1, &vCross2 );

        if( D3DXVec3Dot( &vNormal, &vLightDir ) >= 0 )
        {
          pEdges[2*dwEdgesAmt] = wPolyIndex0;
          pEdges[2*dwEdgesAmt+1] = wPolyIndex1;

          pEdges[2*dwEdgesAmt+2] = wPolyIndex1;
          pEdges[2*dwEdgesAmt+3] = wPolyIndex2;

          pEdges[2*dwEdgesAmt+4] = wPolyIndex2;
          pEdges[2*dwEdgesAmt+5] = wPolyIndex0;

          dwEdgesAmt+=3;
        }
    }
//輪郭としてリストアップされたポリゴンを引き伸ばす（ここで適当に15単位伸ばす）
    for( DWORD i = 0; i < dwEdgesAmt; i++ )
    {
         vVertex1 = pVertices[pEdges[2*i]].Pos;
         vVertex2 = pVertices[pEdges[2*i+1]].Pos;
         vVertex3 = vLightDir*15-vVertex1;
         vVertex4 = vLightDir*15-vVertex2;

        g_pVertices[g_dwVerticesAmt] = vVertex1;
        g_pVertices[g_dwVerticesAmt+1] = vVertex2;
        g_pVertices[g_dwVerticesAmt+2] = vVertex3;

        g_pVertices[g_dwVerticesAmt+3] = vVertex2;
        g_pVertices[g_dwVerticesAmt+4] = vVertex4;
        g_pVertices[g_dwVerticesAmt+5] = vVertex3;
    g_dwVerticesAmt+=6;
    }
    pMesh->UnlockVertexBuffer();
    pMesh->UnlockIndexBuffer();
    delete pEdges;

    return S_OK;
}
```

●ボリュームをステンシルバッファに書き込む関数

```
HRESULT RenderVolumeToStencil(LPDIRECT3DDEVICE9 pDevice,D3DXMATRIX*
pmatWorld)
{
    pDevice->SetRenderState( D3DRS_ZWRITEENABLE, FALSE );
    pDevice->SetRenderState( D3DRS_STENCILENABLE,TRUE );
    pDevice->SetRenderState( D3DRS_SHADEMODE,D3DSHADE_FLAT );

    pDevice->SetRenderState( D3DRS_STENCILFUNC, D3DCMP_ALWAYS );
    pDevice->SetRenderState( D3DRS_STENCILZFAIL,D3DSTENCILOP_KEEP );
    pDevice->SetRenderState( D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP );

    pDevice->SetRenderState( D3DRS_STENCILREF, 0x1 );
    pDevice->SetRenderState( D3DRS_STENCILMASK,  0xffffffff );
    pDevice->SetRenderState( D3DRS_STENCILWRITEMASK, 0xffffffff );
    pDevice->SetRenderState( D3DRS_STENCILPASS, D3DSTENCILOP_INCR );

    pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
    pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_ZERO );
    pDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_ONE );
    //まずは片面をステンシルに書き込む
    pDevice->SetTransform( D3DTS_WORLD, pmatWorld  );
    pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_TEX1  );
      pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST,
g_dwVerticesAmt/3,g_pVertices, sizeof(D3DXVECTOR3) );
    //その裏面もステンシルに書き込む
    pDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_CW );
    pDevice->SetRenderState( D3DRS_STENCILPASS, D3DSTENCILOP_DECR );
    pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_TEX1  );
      pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST,
g_dwVerticesAmt/3,g_pVertices, sizeof(D3DXVECTOR3) );

    // レンダリングステートを元の状態に戻す
    pDevice->SetRenderState( D3DRS_SHADEMODE, D3DSHADE_GOURAUD );
    pDevice->SetRenderState( D3DRS_CULLMODE,  D3DCULL_CCW );
    pDevice->SetRenderState( D3DRS_ZWRITEENABLE,  TRUE );
    pDevice->SetRenderState( D3DRS_STENCILENABLE, FALSE );
    pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, FALSE );
    return S_OK;
}
```

●②と③の処理をしている部分

ステンシルバッファの各ピクセル情報を頼りに、シャドウとなる部分を判定し、シャドウを板ポリゴン上にレンダリング（ポストプロセス）します。

```
VOID RenderShadow(LPDIRECT3DDEVICE9 pDevice)
{
    pDevice->SetRenderState( D3DRS_STENCILREF, 0x1 );
    pDevice->SetRenderState( D3DRS_STENCILFUNC, D3DCMP_LESSEQUAL );
    pDevice->SetRenderState( D3DRS_STENCILPASS, D3DSTENCILOP_KEEP );
    pDevice->SetRenderState( D3DRS_ZENABLE,FALSE );
    pDevice->SetRenderState( D3DRS_STENCILENABLE,TRUE );
    pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,TRUE );
    pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
    pDevice->SetRenderState( D3DRS_DESTBLEND,D3DBLEND_INVSRCALPHA );
    // 半透明板ポリゴンを画面の最前面に描画。この板ポリゴンをステンシルで切り取った部分がシャドウとなる
    struct COLOR_POINT
    {
        D3DXVECTOR4 vecCoord;
        DWORD dwColor;
    };
    BYTE bAlpha=150;
    COLOR_POINT ColorPoint[4]=
    {
        D3DXVECTOR4(0,0,0,1),D3DCOLOR_ARGB(bAlpha,10,10,10),
        D3DXVECTOR4(800,0,0,1),D3DCOLOR_ARGB(bAlpha,10,10,10),
        D3DXVECTOR4(0,600,0,1),D3DCOLOR_ARGB(bAlpha,10,10,10),
        D3DXVECTOR4(800,600,0,1),D3DCOLOR_ARGB(bAlpha,10,10,10)
    };
    pDevice->SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE );
    pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2, ColorPoint, sizeof(COLOR_POINT));
}
```

これまでの説明でなんとなくは分かっても、原理を理解できる人はいないと思います。原理をきちんと知りたい人のために、より細かく段階分けすると次のようになります。

まず、輪郭部分の頂点を光源ベクトル方向に引き伸ばします。それにより影の領域（シャドウボリューム）が出来ます。ここがミソですが、ステンシルバッファー上に影のステンシル（影の型）を作ります。

どうやってステンシルを作るかというと、シャドウボリュームの表と裏を分けてレンダリングします。レンダリングする際にはステンシルバッファを有効にし、ステンシルバッファに値が書き込まれるようにします。

ボリュームの表側をレンダリングするときはステンシルバッファに+1が書き込まれるようにし、ボリュームの裏側をレンダリングするときはステンシルバッファに-1が書き込まれるようにレンダリングステートを設定します。

こうすれば上手い具合に影のステンシルが出来上がります。なぜなら、視点から見てボリュームの表と裏が重なっている場合は、ステンシル値が相殺されてゼロになり、表と裏の間に何らかのポリゴンが割り込んでいる場合にはじめて+1となります。+1となっているということはその部分は影としてレンダリングする部分と判断できます。つまり、ボリュームの中になにもポリゴンがなければステンシル値=0（影はできない）、ボリュームの中に何らかのポリゴンがあればそのポリゴンには影が落ちることになりますが、それをステンシル値=+1で判断できるというものです。

One Point

この技法においてステンシルバッファが重要なファクターとなっていることから、「ステンシルシャドウ法」あるいは「ステンシルシャドウボリューム法」とも呼ばれています。

219 ポイントライト時のシャドーボリューム

ライトの位置を反映させる

初級

サンプルプロジェクト 219 シャドウボリューム（ポイントライト版）

直前のTipsにおけるライトは“ディレクショナルライト（無限遠光源)”でした。ポイントライトやスポットライトのように“位置”を持つライトの場合、直前のサンプルとはコードが多少異なります。と言っても、ShadowVolume.cpp側のコードは全く変更なく使えます。異なるのは、Main.cpp側であって、変更は容易です。

●ディレクショナルライトの場合（Main.cpp側）

```
//ライトを回転
  D3DXMATRIX  mRotation;
  D3DXMatrixRotationY(&mRotation,D3DXToRadian(2));//回転速度は適当に2度/フレーム
  D3DXVec3TransformCoord(&g_vLightDirection,&g_vLightDirection,&mRotation);

CreateVolume(Thing[1].pMesh,vLightDirection)
```

●ポイントライトの場合（Main.cpp側）

```
//現在のポイントライト位置でのシャドウボリュームを作成
 D3DXVECTOR3 vLightDirection=Thing[1].vecPosition -
(D3DXVECTOR3)light.Position;
 D3DXMATRIX matInverse;
 D3DXMatrixInverse( &matInverse, NULL, &Thing[1].matWorld );
 D3DXVec3TransformNormal( &vLightDirection, &vLightDirection, &matInverse );

CreateVolume(Thing[1].pMesh,vLightDirection)
```

CreateVolume関数はメッシュポインターと“ライトの方向“を引数にとります。ポイントライトの場合は、多少手間を掛けて、ライト位置から方向を算出してやる必要があります。

ちなみに、本質ではありませんが、本サン

プルでのライトは固定機能シェーダーでのライティングなので、ポイントライトで照らしている部分が綺麗に丸くなってくれず、ポリゴンの形状に依存して“カクカク”してしまいます。プログラマブルシェーダーによる“ピクセル単位のライティング”（パーピクセルライティング）をすると綺麗になるのですが、コードが複雑になるのでパーピクセルライティングは行っていません。

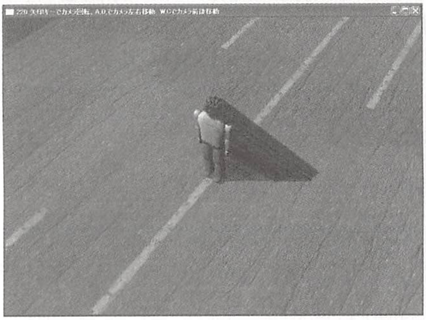
220 シャドウボリューム自体をレンダリングする

●シャドウボリュームがどのようなものなのか可視化して確認する

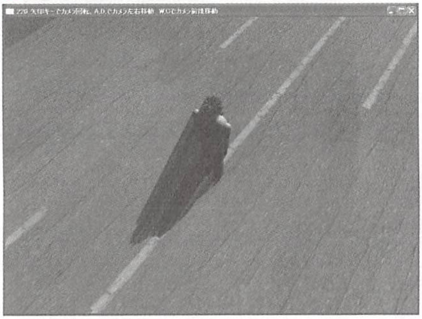
上級

サンプルプロジェクト 220 シャドウボリューム自体のレンダリング

▼実行結果


220 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Cでカメラ前後移動

▼実行結果


220 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Cでカメラ前後移動

シャドウボリュームが“実際にどのような形状になっているのか？”を知りたいと思うはずです。

このサンプルでは、構造自体はTips218と全く同じで、ただボリュームを表示するRenderVolume関数を追加しただけです。

```
VOID RenderVolume( LPDIRECT3DDEVICE9 pDevice,D3DXMATRIX* pmatWorld)
{
     pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1);

     pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,TRUE );
     pDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
     pDevice->SetRenderState( D3DRS_DESTBLEND,D3DBLEND_INVSRCALPHA );

     D3DMATERIAL9 Material;
     ZeroMemory(&Material,sizeof(D3DMATERIAL9));
     Material.Diffuse.r=0.0f;
     Material.Diffuse.g=0.0f;
```

```
    Material.Diffuse.b=6.0f;
    Material.Diffuse.a=0.3f;
    pDevice->SetMaterial(&Material);

    pDevice->SetTransform( D3DTS_WORLD, pmatWorld );

    pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST,
           g_dwVerticesAmt/3,g_pVertices, sizeof(D3DXVECTOR3) );

    pDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,FALSE );
}
```

221 ダブルステンシルでシャドウボリュームを高速化する

●2枚のステンシルバッファに同時に書きこむことにより高速化を図る 上級

サンプルプロジェクト 221 シャドウボリューム（ダブルステンシル版）

Tips218やTips219のコードでは、ステンシルバッファは1つだけなので、レンダリングするパスを2回行う必要がありました。しかし、グラフィックカードがダブルステンシルをサポートしていると、ステンシルバッファを2つ同時に使用するので、レンダリングは1パスで済みます。

つまりTips218やTips219におけるステンシルバッファへの書き込みは、実は遅い（とうか時代遅れの）構造だったのです。

さて、Tips218やTips219と異なるのはステンシルバッファへの書き込み部分、すなわちRenderVolumeToStencil関数だけが異なります。

Tips218やTips219のの場合、

```
//まずは片面をステンシルに書き込む
pDevice->SetTransform( D3DTS_WORLD, pmatWorld  );
pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_TEX1  );
pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST, g_dwVerticesAmt/3,g_pVertices,
sizeof(D3DXVECTOR3) );
//その裏面もステンシルに書き込む
pDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_CW );
pDevice->SetRenderState( D3DRS_STENCILPASS, D3DSTENCILOP_DECR );
pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_TEX1  );
pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST, g_dwVerticesAmt/3,g_pVertices,
sizeof(D3DXVECTOR3) );
```

DrawPrimitiveUPを2回実行している（2パス）のが分かると思います。

それに対し、本サンプルでは1パスで行っています。

```
// 両面同時にステンシルに書き込む
pDevice->SetRenderState( D3DRS_TWOSIDEDSTENCILMODE, TRUE );
pDevice->SetRenderState( D3DRS_CULLMODE,  D3DCULL_NONE );
pDevice->SetRenderState( D3DRS_CCW_STENCILFUNC,   D3DCMP_ALWAYS );
pDevice->SetRenderState( D3DRS_CCW_STENCILZFAIL, D3DSTENCILOP_KEEP );
pDevice->SetRenderState( D3DRS_CCW_STENCILFAIL,   D3DSTENCILOP_KEEP );
pDevice->SetRenderState( D3DRS_CCW_STENCILPASS, D3DSTENCILOP_DECR );

pDevice->SetTransform( D3DTS_WORLD, pmatWorld );
pDevice->SetFVF( D3DFVF_XYZ | D3DFVF_TEX1  );
pDevice->DrawPrimitiveUP( D3DPT_TRIANGLELIST, g_dwVerticesAmt/3,g_pVertices,
sizeof(D3DXVECTOR3) );
```

なお、ダブルステンシルをサポートしているかどうか確認するようにしたほうがいいでしょう。本サンプルでも念のため次のように確認しています。

```
//ダブル・ステンシルがサポートされているか？　ダブルステンシルが使えるとパフォーマンスが上がる
D3DCAPS9  Caps;
pD3d->GetDeviceCaps(  D3DADAPTER_DEFAULT,  D3DDEVTYPE_HAL,  &Caps  );
if(  (  Caps.StencilCaps  &  D3DSTENCILCAPS_TWOSIDED  )  ==  0  )
{
    MessageBox(0,"ダブルステンシルに対応していません","",MB_OK);
    return  E_FAIL;
}
```

222 シャドウマップ法で影を付ける

深度を記録したテクスチャ（シャドウマップ）を参照しながら影となる部分を特定する　上級

サンプルプロジェクト 222 シャドウマップ

シャドウマップ法はTips218シャドウボリューム法と並び、本格的なシャドウを生成できます。本格的というのは、他のジオメトリに影が落ちるのはもちろん、加えてセルフシャドウ（自分の影が自分自身にも落ちる）もかかります。そして、影は不透明ではなく、影が落ちている部分のテクスチャも見ることができます。つまり、平面投影法の欠点を全て解消した完全な影を生成する技法です。

シャドウボリュームは、プログラマブルシ

ェーダーを使用せずとも実装できましたが、このシャドウマップはプログラマブルシェーダーなしに実装は事実上不可能です（1フレームレンダリングするのに数十秒かけてもいいのなら話は別です…）。

また、シャドウボリュームにおいて最終的な影付けはポストプロセスでしたが、シャドウマップにポストプロセスは不要です。

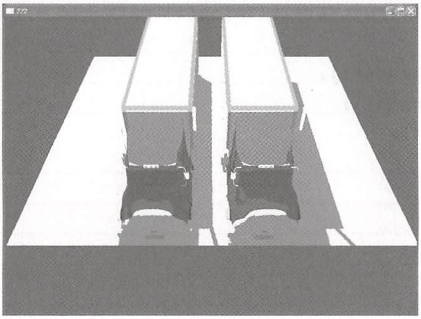
シャドウマップの仕組みは、

①まずプログラマブルシェーダーにより“ライトからの視点”でシーンをレンダリングします。レンダリング先は通常のフレームバッファではなく“テクスチャ上に”レンダリングします。このテクスチャの機能はライトからの各ピクセルの深度（Z値）の参照用です（このテクスチャを深度テクスチャと呼ぶことにします）。

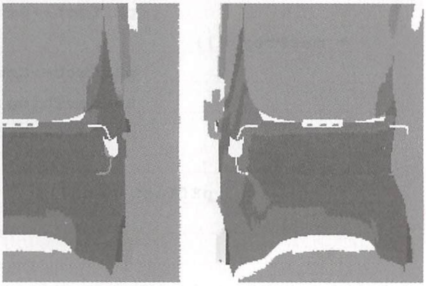
②次に、プログラマブルシェーダーにより“通常のカメラ視点”でシーンをレンダリングします。レンダリング先は通常のフレームバッファ（バックバッファ）です。ただし、深度テクスチャを参照して、深度を比較しながらピクセル色を決定していきます。

それをすることにより影部分を特定でき、影を付けることができます。

▼影以外ではディフューズ色のみ反映



▼セルフシャドウもレンダリングされている



●①の処理をしている部分：アプリケーションコード側

```
LPDIRECT3DSURFACE9 pBackBuffer = NULL; バックバッファ
LPDIRECT3DSURFACE9 pZBuffer = NULL; Zバッファ

LPDIRECT3DTEXTURE9 pDepthTexture=NULL; 深度テクスチャ
LPDIRECT3DSURFACE9 pTexZBuffer = NULL; 深度テクスチャをレンダーターゲットにする際のz
バッファ
```

```
//"シェーダーで"、"シーンを"、"深度用テクスチャ上に"、レンダリング
 {
     LPDIRECT3DSURFACE9 pTexSurf = NULL;
     pDepthTexture->GetSurfaceLevel(0,&pTexSurf);
     pDevice->SetRenderTarget(0,pTexSurf );
     pDevice->SetDepthStencilSurface(pTexZBuffer);
     SAFE_RELEASE(pTexSurf);
     pDevice->SetRenderState( D3DRS_LIGHTING, FALSE );
     pDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
            D3DCOLOR_ARGB(255,255,255,255), 1.0f, 0 );
     pEffect->SetTechnique("tecMakeDepthTexture");

     D3DXMATRIX matView;
     pDevice->GetTransform(D3DTS_VIEW,&matView);
     D3DXMATRIX matProj;
     pDevice->GetTransform(D3DTS_PROJECTION,&matProj);

     pEffect->Begin( NULL, 0 );
     {
            for(INT i=0;i<THING_AMOUNT;i++)
            {
                  pEffect->BeginPass(0);
                  pEffect->SetMatrix("matWLP",&(Thing[i].matWorld * matLight
* matProj) );
                  pEffect->CommitChanges();
                  RenderThing(&Thing[i]);
                  pEffect->EndPass();
                         }
            pEffect->End();
     }
 }
```

●シェーダー側

```
struct VS_OUT
{
    float4 Pos : POSITION;
    float2 DecalTC : TEXCOORD0;
    float4 ShadowTC : TEXCOORD1;
    float4 DepthValue : TEXCOORD2;
};
```

```
float4x4  matWCP;
float4x4  matWLP;
float4x4  matWLPT;

float4  Diffuse;

float  TexWidth=1024;
float  TexHeight=1024;
```

```
texture  DepthTex;
sampler  DepthSampler  =  sampler_state
{
    Texture  =  (DepthTex);
    MinFilter  =  Point;
        MagFilter  =  Point;
        MipFilter  =  NONE;
        AddressU  =  Clamp;
        AddressV  =  Clamp;
};
```

```
/////////////////////////////////////////
//シャドウマップ作成用 バーテックス・シェーダー
/////////////////////////////////////////
VS_OUT VS_Depth(float4 Position : POSITION)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos=mul(Position,matWLP);
    Out.DepthValue=Out.Pos;

    return Out;
}
```

```
/////////////////////////////////////////
//シャドウマップ作成用 ピクセル・シェーダー
/////////////////////////////////////////
float4 PS_Depth( VS_OUT In) : COLOR
{
    float4 FinalColor;

    FinalColor=In.DepthValue.z / In.DepthValue.w;

    return FinalColor;
}
```

●②の処理をしている部分

```
//通常のフレームバッファにシーンをレンダリング（シェーダー内では、同時にシャドウマップをカメラ空間にマッピング）
 {
     D3DXMATRIX mTexSpace;
     ZeroMemory(&mTexSpace,sizeof(D3DXMATRIX));
```

```
    mTexSpace._11=0.5;
    mTexSpace._22=-0.5;
    mTexSpace._33=1;
    mTexSpace._41=0.5;
    mTexSpace._42=0.5;
    mTexSpace._44=1;

    pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1 );
    pDevice->SetRenderTarget(0,pBackBuffer );
    pDevice->SetDepthStencilSurface(pZBuffer);

    pEffect->SetTechnique("tecRenderScene");
    pEffect->SetTexture("DepthTex",pDepthTexture);
    pEffect->Begin( NULL, 0 );
    {
        pEffect->BeginPass(0);
        // レンダリング
        for(INT i=0;i<THING_AMOUNT;i++)
        {
            pEffect->SetMatrix("matWCP",&(Thing[i].matWorld * matView
* matProj) );
            pEffect->SetMatrix("matWLP",&(Thing[i].matWorld * matLight
* matProj) );
            pEffect->SetMatrix("matWLPT",&(Thing[i].matWorld *
matLight * matProj *mTexSpace ) );
            RenderThingWithShader(&Thing[i]);
        }
        pEffect->EndPass();
    }
    pEffect->End();
 }
```

```
////////////////////////////////////////////
//シーンレンダリング用 バーテックス・シェーダー
////////////////////////////////////////////
VS_OUT VS_Scene(float4 Position : POSITION, float3 Normal : NORMAL,float2 TC
: TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos=mul(Position,matWCP);
    Out.DecalTC=TC;
    Out.ShadowTC=mul(Position,matWLPT);//テクスチャ座標をライト空間にマッピング。
    Out.DepthValue=mul(Position,matWLP)-0.5;//影がなくなる部分がでるが縞々が出るよりは…

    return Out;
}
//////////////////////////////////////////
//シーンレンダリング用 ピクセル・シェーダー
//////////////////////////////////////////
float4 PS_Scene( VS_OUT In) : COLOR
{
    float4 FinalColor;

    FinalColor=Diffuse;

    float Shadow=tex2Dproj( DepthSampler, In.ShadowTC ).r;//テクスチャ座標をライト空間にマッピング。

    if(Shadow * In.DepthValue.w<In.DepthValue.z)
    {
        FinalColor=Diffuse/2;
    }

    return FinalColor;
}
```

223 シャドウマップ法で影を付ける(ライティングを加味)

●Tips222のシェーダーにライティングを追加して鏡面反射もレンダリングする 上級

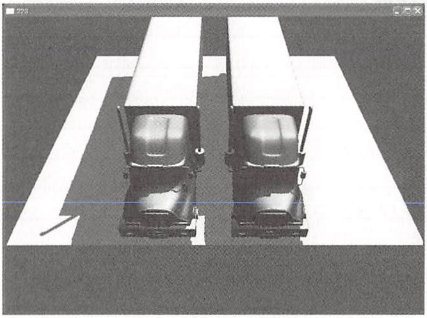
サンプルプロジェクト 223 シャドウマップ（+ライティング）

直前のTipsにライティング計算を追加しました。陰影と鏡面反射が加わり、より品質が向上しています。

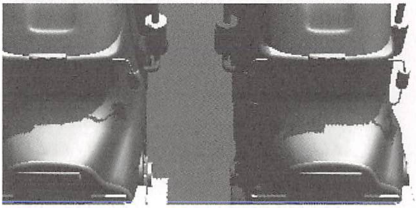
ライティング計算はTips131「プログラマブルシェーダーで光沢を付ける」の計算をそのまま組み込んでいます。

シェーダーコード（ShadowMap.fx）で変更したのは、シーンレンダリングの頂点シェーダーとピクセルシェーダーだけで、深度テクスチャ作成の頂点シェーダー、ピクセルシェーダーは変更していません。

▼シェーダーに鏡面反射ライティング項も追加



▼セルフシャドウもレンダリングされている



```
///////////////////////
//型定義
///////////////////////
struct VS_OUT
{
    float4 Pos : POSITION;
    float3 Light : TEXCOORD0;
    float3 Normal : TEXCOORD1;
    float3 View : TEXCOORD2;
    float2 DecalTC : TEXCOORD3;
    float4 ShadowTC : TEXCOORD4;
    float4 DepthValue : TEXCOORD5;
};

/////////////////////////////////////////
//グローバル
/////////////////////////////////////////
float4x4 matW;
float4x4 matWCP;
float4x4 matWLP;
float4x4 matWLPT;

float4 Diffuse;
float4 vecLightDir;
float4 vecEye;
```

```
/////////////////////////////////////////
//シーンレンダリング用 バーテックス・シェーダー
/////////////////////////////////////////
VS_OUT VS_Scene(float4 Position : POSITION, float3 Normal : NORMAL,float2
: TEXCOORD0)
```

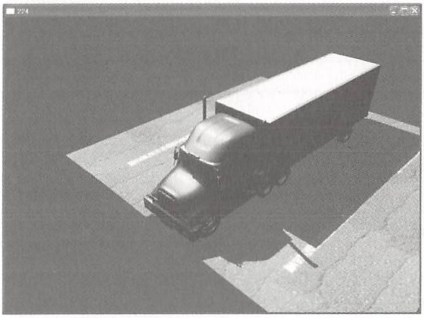
```
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos=mul(Position,matWCP);
    Out.Light = vecLightDir;
    Out.View = vecEye -normalize(mul(Position, matW));
    Out.Normal = mul(Normal, matW);
    Out.DecalTC=TC;
    Out.ShadowTC=mul(Position,matWLPT);//テクスチャ座標をライト空間にマッピング。
    Out.DepthValue=mul(Position,matWLP)-0.5;//影がなくなる部分がでるが縞々が出るよりは…

    return Out;
}
////////////////////////////////////////
//シーンレンダリング用 ピクセル・シェーダー
////////////////////////////////////////
float4 PS_Scene( VS_OUT In) : COLOR
{
    float4 FinalColor;

    float3 Normal = normalize(In.Normal);
    float3 LightDir = normalize(In.Light);
    float3 ViewDir = normalize(In.View);

    float4 NL = saturate(dot(Normal, LightDir));
    float3 Reflect = normalize(2 * NL * Normal - LightDir); //反射ベクトルの公式通り
    float4 Specular = pow(saturate(dot(Reflect, ViewDir)), 4);


    FinalColor=Diffuse * NL + Specular; //影の部分でなければ、この時点のカラーが最終カラー


    float Shadow=tex2Dproj( DepthSampler, In.ShadowTC ).r;//テクスチャ座標をライト空間にマッピング。

    if(Shadow * In.DepthValue.w<In.DepthValue.z)
    {
        FinalColor/=4;//影ならば色値を半分にする（もっと暗くしたい場合は2以上で割る）
    }

    return FinalColor;
}
```

224 シャドウマップ法で影を付ける（ライティング、テクスチャを加味）

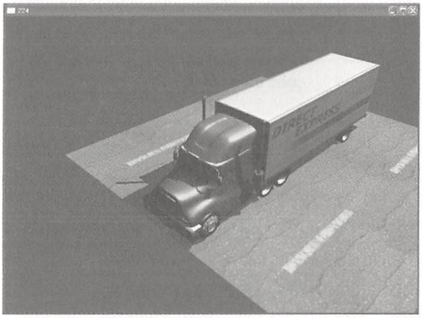
●Tips222のシェーダーにライティングとテクスチャマッピング計算を追加する　上級

サンプルプロジェクト　224　シャドウマップ（＋ライティング、テクスチャ）

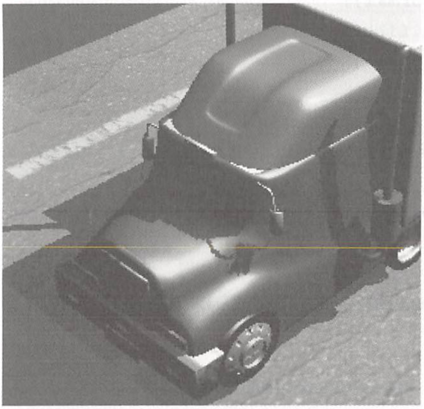
▼実行結果



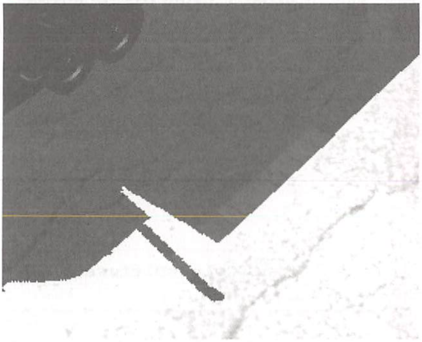
▼シャドウ、ライティング、テクスチャ全てレンダリング



▼セルフシャドウ



▼影が落ちているメッシュのテクスチャも見える



図を見ると、まるでオフラインレンダリング（3DCGソフトでじっくり時間をかけたレンダリング）のような品質です。本サンプルではシェーダー内でシャドウィングとライティング、さらにテクスチャマッピングも行っています。

```
////////////////////////
//型定義
////////////////////////
struct VS_OUT
```

```
{
    float4 Pos : POSITION;
    float3 Light : TEXCOORD0;
    float3 Normal : TEXCOORD1;
    float3 View : TEXCOORD2;
    float2 DecalTC : TEXCOORD3;
    float4 ShadowTC : TEXCOORD4;
    float4 DepthValue : TEXCOORD5;
};
//////////////////////
//グローバル
//////////////////////
float4x4 matW;
float4x4 matWCP;
float4x4 matWLP;
float4x4 matWLPT;

float4 Diffuse;
float4 vecLightDir;
float4 vecEye;

float TexWidth=1024;
float TexHeight=1024;

texture DepthTex;
sampler DepthSampler = sampler_state
{
    Texture = (DepthTex);
    MinFilter = Point;
    MagFilter = Point;
    MipFilter = NONE;
    AddressU = Clamp;
    AddressV = Clamp;
};

texture DecalTexture;
sampler DecalSampler = sampler_state
{
    Texture = (DecalTexture);
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = NONE;
    AddressU = Clamp;
    AddressV = Clamp;
};
```

```
//////////////////////////////////////////
//シーンレンダリング用 バーテックス・シェーダー
//////////////////////////////////////////
VS_OUT VS_Scene(float4 Position : POSITION, float3 Normal : NORMAL,float2 TC
: TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;

    Out.Pos=mul(Position,matWCP);
    Out.Light = vecLightDir;
    Out.View = vecEye -normalize(mul(Position, matW));
    Out.Normal = mul(Normal, matW);
    Out.DecalTC=TC;
    Out.ShadowTC=mul(Position,matWLPT);//テクスチャ座標をライト空間にマッピング。
    Out.DepthValue=mul(Position,matWLP)-0.5;//影がなくなる部分がでるが縞々が出るよ
りは…

    return Out;
}
//////////////////////////////////////////
//シーンレンダリング用 ピクセル・シェーダー
//////////////////////////////////////////
float4 PS_Scene( VS_OUT In) : COLOR
{
    float4 FinalColor;

    float3 Normal = normalize(In.Normal);
    float3 LightDir = normalize(In.Light);
    float3 ViewDir = normalize(In.View);

    float4 NL = saturate(dot(Normal, LightDir));
    float3 Reflect = normalize(2 * NL * Normal - LightDir); //反射ベクトルの公
式通り
    float4 Specular = pow(saturate(dot(Reflect, ViewDir)), 4);


    FinalColor=(Diffuse+tex2D(DecalSampler ,In.DecalTC) )* NL + Specular; //
影の部分でなければ、この時点のカラーが最終カラー


    float Shadow=tex2Dproj( DepthSampler, In.ShadowTC ).r;//テクスチャ座標をラ
イト空間にマッピング。

    if(Shadow * In.DepthValue.w<In.DepthValue.z)
    {
            FinalColor/=4;//影ならば色値を半分にする(もっと暗くしたい場合は2以上で割る)
    }

    return FinalColor;
}
```

225 深度テクスチャを表示してみる

●深度テクスチャを可視化してピクセルがどのようなものなのかを見る 上級

サンプルプロジェクト 225 深度テクスチャを見てみる

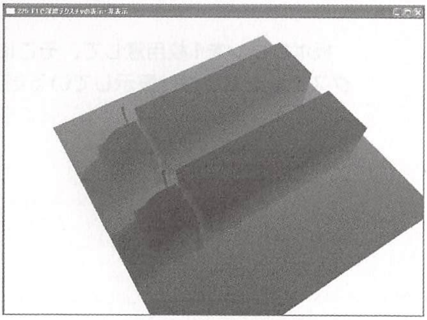
深度テクスチャも目的はシャドウ部分決定のためにシェーダー内部で参照することなので表示する必要がありません。しかし、やはり深度テクスチャが実際どのような陰影になっているのか、とか、深度テクスチャがどのようなアングルになっているのかを見てみたいと思うはずです。

深度テクスチャを表示してみると、そこに映るシーンが回転しているのがわかります。これは本サンプル（本書すべてのシャドウマップサンプルで）でライトを回転させているからです。思い出してください、深度テクスチャに格納するデータは“ライトから見たシーン”ということを。そのため本来のシーンが回転していなくても深度テクスチャ上のシーンは回転することになります。

また、ライトを回転させなくとも、通常はライトからの視点とカメラからの視点は異なるので、最終的なレンダリングと深度テクスチャ上のシーンのアングルは異なります。深度テクスチャ上の明暗の差は、ライトからの距離（Z深度）の差です。ライトに近いほど黒くなり、ライトから遠いほど白くなります。

本サンプルは、最もコードが少ないTips222のサンプルコードに表示ルーチンを加えたものです。と言っても深度テクスチャの作成部分は全てのシャドウマップサンプルで同一です。したがって深度テクスチャの中身自体は全てのサンプルで同じです。表示させるために付け加えるコードは、全てMain.cpp側であって、シェーダーコードは一切変更不要です。

▼実行結果



```
struct   TEX_VERTEX
{
    FLOAT   x,y,z;
    FLOAT   u,v;
};

//シャドウマップを表示
 if(boShowDepthTex)
 {
   pDevice->SetRenderTarget(0,pBackBuffer  );
   pDevice->SetDepthStencilSurface(pZBuffer);

   struct   TEX_VERTEX
  {
     D3DXVECTOR4   vCoord;
     D3DXVECTOR2   vTexCoord;
  };

  TEX_VERTEX   TexPoint[4]=
  {
     D3DXVECTOR4(0,0,0,1),D3DXVECTOR2(0,0),
     D3DXVECTOR4(800,0,0,1),D3DXVECTOR2(1,0),
     D3DXVECTOR4(0,600,0,1),D3DXVECTOR2(0,1),
     D3DXVECTOR4(800,600,0,1),D3DXVECTOR2(1,1)
  };
  pDevice->SetTexture(0,pDepthTexture);
  pDevice->SetFVF(D3DFVF_XYZRHW  |  D3DFVF_TEX1  );
  pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP,2,              TexPoint,
sizeof(TEX_VERTEX));
 }
```

板ポリゴンを1枚用意して、そこに深度テクスチャを貼り付け表示しているだけです。

当然、レンダーターゲットは通常のバックバッファにしなければなりません。

第 2 章

226～251

ダイナミクスの極意

2-1　レイによる衝突判定（226～240）
2-2　バウンディングスフィアによる衝突判定（241～243）
2-3　バウンディングボックスよる衝突判定（244～249）
2-4　落下と跳ね返り（250～251）

直線と面の交差検出

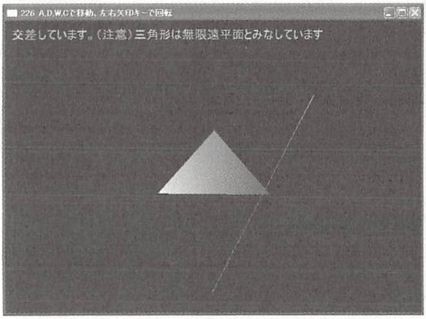
226 直線と無限遠平面の交差を検出する

●パラメトリック方程式によりシンプルに交差を検出する

中級

サンプルプロジェクト　226　直線と無限遠平面

▼実行結果



交差しています。(注意)三角形は無限遠平面とみなしています

本章での直線は厳密に言えば線分です。線分とは直線のうち、両方に端点があるものです。線分と平面の交差判定にはパラメトリック方程式を使用しますが、端点が無いと上手く機能しません（というか式が書けません）。

さて、線分上の任意の点は両端点の間にありますから、次のような式で表せます。

線分の端点をP0(x0,y0,z0)とP1(x1,y1,z1)とすると、線分上の任意の点P(x,y,z)は、媒介変数（パラメーター）tと端点同士の方向ベクトルV(Vx,Vy,Vz)を用いて次のように書けます。

P=P0+tV
(但し　0<=t<=1、V=P1−P0)

したがって、Pの3成分はそれぞれ次のようになります。

X=x0+t*Vx
Y=y0+t*Vy
Z=z0+t*Vz;
(但し　0<=t<=1)

tがゼロの時は始点の端点（P0）と重なり、tが1のときは、終点の端点(P1)と重なります。tがゼロと1の間では、始点と終点の間の点を表すのですから、すなわち線分を表しているというわけです。

パラメーターを用いたこのような表現をパラメトリック方程式と言います。線分をパラメーター表示することにより、シンプルなコードで交点を求めることができます。

この方程式をどのように利用するかと言うと、まずパラメーター表示した線分上の点を平面の平面方程式ax+by+cz+d=0に代入します。

A(x0+t*vx)+b(x0+t*vx)+c(x0+t*vx)+d=0

これを、tについて解きます。

t=−(a*x1+b*y1+c*z1+d)/(a*vx+b*vy+c*vz)

ベクトルV(vx,vy,vz)は、端点から求めたものなので、次のように完全に端点と平面方程式の係数のみで表すことが出来ます。

t=−(a*x1+b*y1+c*z1+d)/(a*(x0−x1)+b*(y0−y1)+c*(z0−z1))…

……①

多少変形すると、次のようにも書けます。

t=−(a＊x1+b＊y1+c＊z1+d)/((a＊x0+b＊y0+c＊z0)−(a＊x1+b＊y1+c＊z1))………②

ここまで式が導出できれば、あとは簡単です。tを計算して、tがゼロから1の範囲外の場合は交差していない、tがゼロ以上1以下の場合は交差しているということになります。その理由はおわかりでしょう、tがセロから1の範囲外ということは、線分の範囲外ということなのですから、交差しているわけがありません。

コードでは、②式の形をそのまま使用しています。

```
Main.h
struct  RAY  //レイ
{
    D3DXVECTOR3  vPoint[2];
    D3DXVECTOR3  vDirection;
    D3DXVECTOR3  vPos;
    FLOAT  fYaw;

    RAY()
    {
           ZeroMemory(this,sizeof(RAY));
    }
};

struct  POLYGON  //平面（平面と言っても、無限遠平面上の3頂点を持っているに過ぎない）
{
    D3DXVECTOR3  vPoint[3];
    POLYGON()
    {
           ZeroMemory(this,sizeof(POLYGON));
    }
};
```

●Intersect.cpp

```
BOOL Intersect(RAY＊ pRay,POLYGON＊ pPoly)
{
    D3DXVECTOR3 vP[2];

    //線分（レイ）の端点を得る
    {
           vP[0]=pRay->vPoint[0];
           vP[1]=pRay->vPoint[1];
```

```
        D3DXMATRIX mRotation;
        D3DXMatrixRotationY(&mRotation,pRay->fYaw);
        D3DXVec3TransformCoord(&vP[0],&vP[0],&mRotation);
        D3DXVec3TransformCoord(&vP[1],&vP[1],&mRotation);
        vP[0]+=pRay->vPos;
        vP[1]+=pRay->vPos;
    }
    //判定部分
    {
        D3DXPLANE p;
        D3DXPlaneFromPoints(&p,&pPoly->vPoint[0],&pPoly->vPoint[1],&pPoly-
>vPoint[2]);
        //パラメトリック方程式の媒介変数” t "を解く。
        FLOAT t=- ( ( p.a * vP[1].x) + (p.b*vP[1].y)+(p.c*vP[1].z)+p.d
) /
        ( ( (p.a*vP[0].x)+(p.b*vP[0].y)+(p.c*vP[0].z) ) - ( (p.a*
vP[1].x)+(p.b*vP[1].y)+(p.c*vP[1].z) ) );
        // t が0から1の間であるなら交差していることになる
        if(t>=0 && t<=1.0)
        {
            return TRUE;
        }
    }
    return FALSE;
}
```

ここでの平面は、範囲が無限である「無限遠平面」です。ですので、見た目、三角形の外側を横切っていてもレイが交差することになります。無限遠平面と交差しない状況は、両方の端点が無限遠平面の外側にあるときだけです。

227 平面方程式を求める

空間上の任意の3点から平面方程式を導く

中級

サンプルプロジェクト　227 平面方程式

コード上で平面を表現するにはどうしたらいいのでしょうか？

もちろん視覚的にはポリゴンで可視化するわけですが、計算においては“平面方程式”により表現されます。例えばDirect3Dで定義されているD3DXPLANEという平面構造体は、aX＋bY＋cZ＋d＝0という一般平面方程式の係数から成っています。

```
typedef struct D3DXPLANE
{
    FLOAT a;
    FLOAT b;
    FLOAT c;
    FLOAT d;
} D3DXPLANE;
```

平面方程式が表す平面は、無限の面積をもつ無限遠変面です。いわゆる一般的な閉じた平面を表現するには、平面の内と外を区別する何らかの範囲が必要です。平面方程式には、そのような範囲を意味する係数はありません。また、D3DXPLANE構造体を見ても分るように、面の範囲を表すようなメンバー変数はありません、面の傾きを表す3つのFLOAT値（a,b,c）と原点からの距離を表すFLOAT値(d)が1つあるだけです。無限というと特別な意味を感じるかも知れませんが、一般的に方程式で現われるジオメトリは無限遠です。これは、直線でも同じことです、お馴染みの直線方程式y＝ax＋bが表すのは、無限遠直線(傾きと原点からの距離)であって、両端点がある線分ではありません。線分を表現するには、何らかの範囲情報が必要です。

これは次のTipsで解説する、交点の面内外判定をしなければならない理由にもなります。

さて、直前のTipsにおいて平面方程式はD3DXヘルパー関数のD3DXPlaneFromPoints関数を使用していました。ここでは、その関数を使用してしまっては意味がないので、自前で平面方程式を導きましょう。

D3DXPlaneFromPointsは任意の3点から平面方程式を求めます。本Tipsの自前関数も引数は同じようにしています。

流れは、まず与えられた3点をもとに、3角形の2辺を導きます。その2辺の外積を計算すれば、平面の法線ベクトルを得ます。法線ベクトルの成分は平面方程式の各係数に対応しているので、この時点で平面方程式ax＋by＋cz＋d＝0の係数a,b,cが一気に得られる形になります。

あと残るはスカラー値dですが、dは、平面内の任意の点をPとすると、次のように求めることが出来ます。

d＝－(a＊PのX成分＋b＊PのY成分＋c＊P

のZ成分）

理論を、そのまま素直に分りやすくコードにしたので、実行速度は改善の余地があります。高速化は読者にお任せします。

```
VOID CalcPlane(D3DXPLANE＊ pPlane,POLYGON＊ pPoly)
{
	//辺ベクトル
	D3DXVECTOR3 vecAB,vecBC;
	vecAB=pPoly->vPoint[1]-pPoly->vPoint[0];
	vecBC=pPoly->vPoint[2]-pPoly->vPoint[1];
	D3DXVec3Normalize(&vecAB,&vecAB);
	D3DXVec3Normalize(&vecBC,&vecBC);
	//平面法線ベクトル
	D3DXVECTOR3 vecNormal;
	D3DXVec3Cross(&vecNormal,&vecAB,&vecBC);
	// 法線は平面の傾きでもあるので、そのまま代入
	pPlane->a=vecNormal.x;
	pPlane->b=vecNormal.y;
	pPlane->c=vecNormal.z;
	// d を計算
	pPlane->d=-(pPlane->a＊pPoly->vPoint[0].x + pPlane->b＊pPoly->vPoint[0].y
+ pPlane->c＊pPoly->vPoint[0].z);
}
```

Column

“d”の意味は？

平面方程式ax+by+cz+d=0のa,b,cがそれぞれx,y,zの係数になることから、それらが面の傾きを表すことは、なんとなく分ると思います。では、dは何を表しているのでしょうか？

もし、”d”が無ければ（ゼロであれば）、その平面が原点を通ることを意味します。“d”は、その平面の“原点からの距離”です。別の言い方をすれば、“その平面と平行で、かつ、原点を通る平面“からのシフト量（距離）とも言えます。

228 交点が平面の内か外かを検出する

内積と外積を利用して内外判定を行う

中級

サンプルプロジェクト　229 直線と有限遠平面

　無限遠平面とレイの交差を検出できるようになったら、次に交点が特定の範囲の内にあるか外にあるかの判定を加えれば、有限遠平面との完全な交差判定ができます。

なぜ内外判定が必要なのでしょう？

　これまでのTipsのサンプルを実行すれば分かると思いますが、無限に広がり範囲を持たない平面との交点なので、交点は意図した有限平面の面内には無い場合もあります。これまでのサンプルでは三角形の外側をレイが横切っている場合でも交差となってしまいます。もちろんこれは、無限の平面なので画面上には見えない平面が広がっているからであり、計算上は正しい結果です。

内積と外積の解説はかなり長くなるので割愛します。

　なお、拙書『はじめての3Dゲーム開発』（工学社）でこのあたりの詳細な解説を行っています。

内外判定の関数

```
//BOOL IsInside
// 交点が面の内にあるときはTRUEを、外にあるときはFALSEを返す
BOOL IsInside(D3DXVECTOR3＊ pvecI,D3DXVECTOR3＊ pvecA,D3DXVECTOR3＊ pvecB,
              D3DXVECTOR3＊ pvecC)
{
    //辺ベクトル
    D3DXVECTOR3 vecAB,vecBC,vecCA;
    vecAB=＊pvecB-＊pvecA;
    vecBC=＊pvecC-＊pvecB;
    vecCA=＊pvecA-＊pvecC;
    //辺ベクトルと「頂点から交点へ向かうベクトル」との、それぞれの外積用
    D3DXVECTOR3 vecCrossAB,vecCrossBC,vecCrossCA;
    //「外積結果のベクトル」と平面法線ベクトルとの、それぞれの内積用
    FLOAT fAB,fBC,fCA;
    // 法線用
    D3DXVECTOR3 vecNormal;
    // まず、3頂点から平面方程式を求める。これは、同時に平面の法線を求めることでもある
    D3DXPLANE pln;
    D3DXPlaneFromPoints(&pln,pvecA,pvecB,pvecC);
    vecNormal.x=pln.a;//法線のx成分は平面方程式のx係数
    vecNormal.y=pln.b;//法線のy成分は平面方程式のy係数
    vecNormal.z=pln.c;//法線のz成分は平面方程式のz係数
    D3DXVec3Normalize(&vecNormal,&vecNormal);

    // 各頂点から交点Iに向かうベクトルをvecVとする
    D3DXVECTOR3 vecV;
    // 辺ABベクトル（頂点Bベクトル-頂点Aベクトル)と、頂点Aから交点Iへ向かうベクトル、の外積
を求める
    vecV=＊pvecI-＊pvecA;
    D3DXVec3Cross(&vecCrossAB,&vecAB,&vecV);
    // 辺BCベクトル（頂点Cベクトル-頂点Bベクトル)と、頂点Bから交点Iへ向かうベクトル、の外積
を求める
    vecV=＊pvecI-＊pvecB;
    D3DXVec3Cross(&vecCrossBC,&vecBC,&vecV);
    // 辺CAベクトル（頂点Aベクトル-頂点Cベクトル)と、頂点Cから交点Iへ向かうベクトル、の外積
を求める
    vecV=＊pvecI-＊pvecC;
    D3DXVec3Cross(&vecCrossCA,&vecCA,&vecV);
    // それぞれの、外積ベクトルとの内積を計算する
    fAB=D3DXVec3Dot(&vecNormal,&vecCrossAB);
    fBC=D3DXVec3Dot(&vecNormal,&vecCrossBC);
    fCA=D3DXVec3Dot(&vecNormal,&vecCrossCA);

    // 3つの内積結果のうち、一つでもマイナス符号のものがあれば、交点は外にある。
    if(fAB>=0 && fBC>=0 && fCA>=0)
    {
        // 交点は、面の内にある
        return TRUE;
    }
    // 交点は面の外にある
    return FALSE;
}
```

229 直線と有限遠平面の交差を検出する

●内外判定を追加し、見た目を一致した完全な交差判定を行う

中級

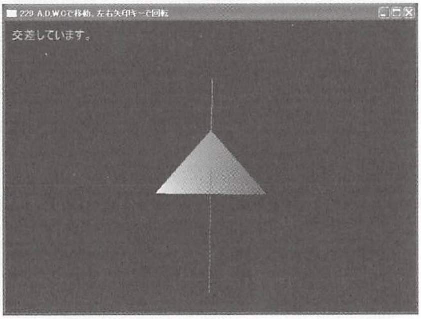
サンプルプロジェクト　229 直線と有限遠平面

レイと無限遠平面の交差判定に、内外判定を追加し、見た目と一致した交差判定を行います。

図を見ると明らかですが、このサンプルでは、

三角形の外にレイがある場合をちゃんと非交差と判定します。

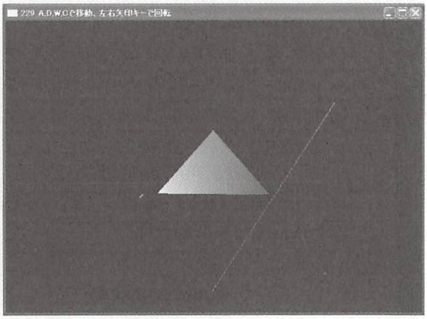
▼実行結果


229 A,D,W,Cで移動、左右矢印キーで回転

交差しています。

このサンプルはTips226とTips228を組み合わせたものです。

▼実行結果


229 A,D,W,Cで移動、左右矢印キーで回転

230 直線と平面の交点座標を算出する

●交差している点の座標を求める

中級

サンプルプロジェクト　230 直線と平面の交点座標

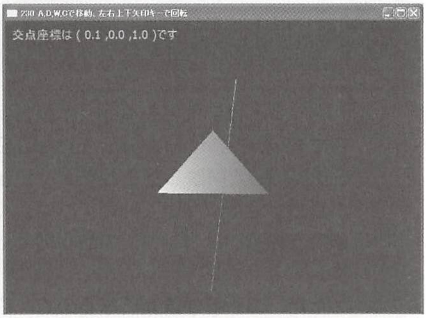
パラメトリック方程式のパラメーター“t”が求まっていれば、始点と終点にtを掛けることにより簡単に交点を特定することができます。

始点をP0(x0,y0,z0)、終点をP1(x1,y1,z1)とすると、交点V(Vx,Vy,Vz)は$V=t*P0+(1-t)*P1$
となります。

したがって、成分は
$Vx=t*x0+(1-t)*x1$
$Vy=t*y0+(1-t)*y1$
$Vz=t*z0+(1-t)*z1$
となります。

コードでの計算は、この計算そのものです。

▼実行結果


230 A,D,W,Cで移動、左右上下矢印キーで回転

交点座標は (0.1 ,0.0 ,1.0)です

●Intersect関数内

```
if(t>=0 && t<=1.0)
{
        //交点座標を得る　tが分かっていれば両端点から簡単に求まる
        D3DXVECTOR3 v;
        v.x = t*vP[0].x+(1-t)*vP[1].x;
        v.y = t*vP[0].y+(1-t)*vP[1].y;
        v.z = t*vP[0].z+(1-t)*vP[1].z;
        pRay->vIntersection=v;
        //交点が三角形の内か外かを判定　（ここで内部となれば、完全な交差となる）
        if(IsInside(&v,&pPoly->vPoint[0],&pPoly->vPoint[1],&pPoly-
>vPoint[2])==TRUE)
        {
                return TRUE;
        }
        return FALSE;
}
```

直線とメッシュの交差検出

231 直線（レイ）とメッシュの交差を検出する

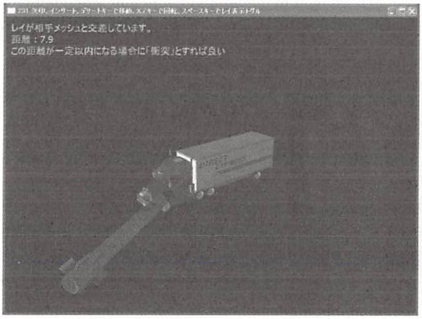
●D3DXIntersect関数を利用してメッシュとの交差判定を行う　中級

サンプルプロジェクト　231　メッシュでのレイ判定

メッシュといえども平面（ポリゴン）の集まりに過ぎませんから、Tips226からTips230で解説したことをメッシュ内部のポリゴン全てについて判定すればよく、これまでとの違いはポリゴンの数が増えただけです。

ただ、コードが煩雑になるので、ここでは便利なD3DXIntersect関数を使用します。D3DXIntersect関数はメッシュとレイの交差を検出してくれるD3DX関数ですが、内部では平面とレイの判定をメッシュ内の各ポリゴンについて行っているだけです。

▼実行結果


231 矢印、インサート、デリートキーで移動、X/Zキーで回転、スペースキーでレイ表示トグル

```
レイが相手メッシュと交差しています。
距離：7.9
この距離が一定以内になる場合に「衝突」とすれば良い
```

```
BOOL RayIntersect(THING＊ pThingA ,THING＊ pThingB,FLOAT＊ pfDistance)
{
    BOOL boHit=FALSE;
    D3DXMATRIX matWorld,matRotation;
    D3DXVECTOR3 vecStart,vecEnd,vecDirection;
    //レイを出すメッシュの位置・回転をレイに適用
    vecStart=pThingA->vecPosition;
    vecEnd=D3DXVECTOR3(0,0,1);
    D3DXMatrixRotationY(&matRotation,pThingA->fYaw);
    D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matRotation);
    vecEnd+=vecStart;

    vecDirection=vecEnd-vecStart;
    D3DXIntersect(pThingB->pMesh,&vecStart,&vecDirection,
&boHit,NULL,NULL,NULL, pfDistance,NULL,NULL);
    if(boHit)
    {
            return TRUE;
    }
    return FALSE;
}
```

232 “ワールド系での”直線(レイ)とメッシュの交差を検出する

●ワールド座標レベルでレイを対象メッシュに正しく当てる

中級

サンプルプロジェクト 232 メッシュでのレイ判定（完全版）

実は、直前のサンプルは不十分です。どこが不十分かと言うと、レイを当てるメッシュ（ここではトラックメッシュ）がワールド系に配置される際、すこしでも移動、回転、スケーリングを行っていると、レイが正しく当たりません。

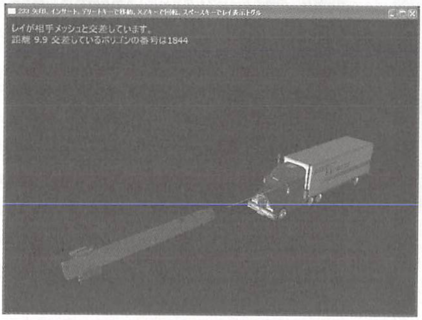
これはD3DXIntersect関数の仕様に起因した結果です。というのもD3DXIntersect関数は対象メッシュの“ローカル系”でレイ判定をするからです。それを回避するために直前のTipsのトラックメッシュがローカル系とワールド系で座標を一致させたのも実は意図的なものです。

このサンプルではトラックメッシュは、ワールド変換によりローカル座標とは異なる座標にマッピングされるので、直前のコードでは画面上レイが当たっているのにも関らず交差を検出できなくなります。

一般的にメッシュはワールド変換でローカル座標とは異なることがほとんどですから、コードは本サンプルのようにしなくてはなりません。

D3DXIntersectによりワールド座標系で完全にレイを正しくあてるためには、レイ側を対象メッシュのローカル座標系に変換することで実現します。具体的には、対象メッシュのワールド変換行列の逆行列をレイに掛けると上手くいきます。

▼実行結果


233 矢印、インサート、デリートキーで移動、XZキーで回転、スペースキーでレイ表示トグル

レイが相手メッシュと交差しています。
距離 9.9 交差しているポリゴンの番号は1844

```
BOOL RayIntersect(THING* pThingA ,THING* pThingB,FLOAT* pfDistance)
{
    BOOL boHit=FALSE;
    D3DXMATRIX matWorld,matRotation;
    D3DXVECTOR3 vecStart,vecEnd,vecDirection;
    //レイを出すメッシュの位置・回転をレイに適用
    vecStart=pThingA->vecPosition;
    vecEnd=D3DXVECTOR3(0,0,1);
    D3DXMatrixRotationY(&matRotation,pThingA->fYaw);
    D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matRotation);
    vecEnd+=vecStart;
    // レイを当てるメッシュが動いていたり回転している場合でも対象のワールド行列の逆行列を用い
れば正しくレイが当たる
    D3DXMatrixTranslation(&matWorld,pThingB->vecPosition.x,pThingB-
>vecPosition.y,pThingB->vecPosition.z);
    D3DXMatrixInverse(&matWorld,NULL,&matWorld);
    D3DXVec3TransformCoord(&vecStart,&vecStart,&matWorld);
    D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matWorld);

    vecDirection=vecEnd-vecStart;
    D3DXIntersect(pThingB->pMesh,&vecStart,&vecDirection,&boHit,NULL,
NULL,NULL, pfDistance,NULL,NULL);
    if(boHit)
    {
        return TRUE;
    }
    return FALSE;
}
```

233 交差しているメッシュ内のポリゴンを得る

●メッシュ内部でレイが当たっているポリゴンを特定する

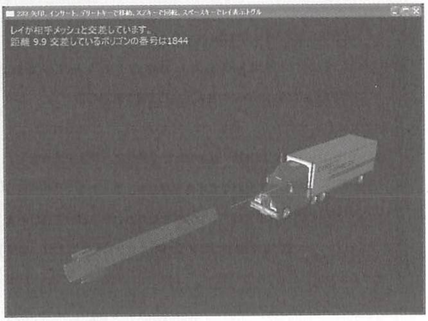
中級

サンプルプロジェクト　233 交差しているメッシュ内ポリゴン

メッシュ内でどのポリゴンがレイと当たっているのか、それを知ることは至極簡単です。簡単よりうよりも、D3DXIntersect関数を実行すればそのポリゴン番号を返してくれるので、なにも特別なことをする必要がありません。

D3DXIntersect関数の第5引数は、当たっているポリゴンの番号を意味し、関数から戻った時、ポリゴンのインデックスが格納されて帰ってきます。

▼実行結果


233 矢印、インサート、デリートキーで移動、ZXキーで回転、スペースキーでレイ表示トグル

レイが相手メッシュと交差しています。
距離 9.9 交差しているポリゴンの番号は1844

●Intersect.cpp側

```
BOOL RayIntersect(THING＊ pThingA ,THING＊ pThingB,FLOAT＊ pfDistance,DWORD＊
pdwIndex)
{
    BOOL boHit=FALSE;
    D3DXMATRIX matWorld,matRotation;
    D3DXVECTOR3 vecStart,vecEnd,vecDirection;
    //レイを出すメッシュの位置・回転をレイに適用
    vecStart=pThingA->vecPosition;
    vecEnd=D3DXVECTOR3(0,0,1);
    D3DXMatrixRotationY(&matRotation,pThingA->fYaw);
    D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matRotation);
    vecEnd+=vecStart;
    // レイを当てるメッシュが動いていたり回転している場合でも対象のワールド行列の逆行列を用い
れば正しくレイが当たる
    D3DXMatrixTranslation(&matWorld,pThingB->vecPosition.x,pThingB-
>vecPosition.y,pThingB->vecPosition.z);
    D3DXMatrixInverse(&matWorld,NULL,&matWorld);
    D3DXVec3TransformCoord(&vecStart,&vecStart,&matWorld);
    D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matWorld);

    vecDirection=vecEnd-vecStart;
```

```
        D3DXIntersect(pThingB->pMesh,&vecStart,&vecDirection,&boHit,
pdwIndex,NULL,NULL, pfDistance,NULL,NULL);
        if(boHit)
        {
                return TRUE;
        }
        return FALSE;
}
```

●Main.cpp側

```
DWORD dwIndex=0;
  if( RayIntersect(&Thing[1],&Thing[0],&fDistance,&dwIndex) &&
fDistance<=RAY_DISTANCE )
  {
        TCHAR szStr[MAX_PATH+1];
        sprintf(szStr,"レイが相手メッシュと交差しています。¥n距離 %3.1f 交差しているポリゴ
ンの番号は%d",fDistance,dwIndex);
        RenderString(szStr,10,10);
  }
```

234 交差しているメッシュ内ポリゴンの平面方程式を計算する

●メッシュ内ポリゴンを構成する3つの頂点を抽出して平面方程式を導く　中級

サンプルプロジェクト　234　ポリゴンの平面方程式

任意の3頂点があれば、平面方程式を作成することができます。

逆に言うと、ポリゴンを構成している3つの頂点を知ることが必要です。D3DXIntersect関数はポリゴンのインデックスを返してはくれすが、そのポリゴンの頂点までは教えてくれません。

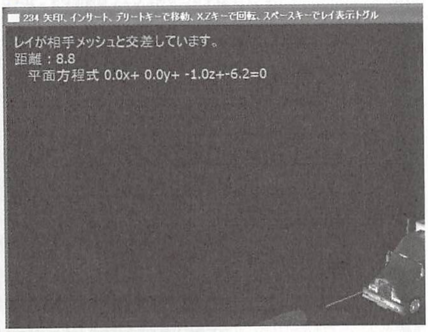
ですので、自前に関数を作成します。本サンプルではFindVerticesOnPoly関数がそれです。

FindVerticesOnPoly関数は、メッシュのインデックスバッファと頂点バッファをロックし、与えられたインデックスであるポリゴンの3頂点を見つけます。

RayIntersect関数内では、FindVerticesOnPoly関数から返された3つの頂点から平面方程式を求めています。

ここでは、簡単化のため、D3DX系関数のD3DXPlaneFromPoints関数を利用しています（この関数を使用せずとも、Tips227の手法により平面方程式は作成できます）。

▼実行結果



234 矢印、インサート、デリートキーで移動、X,Zキーで回転、スペースキーでレイ表示トグル

```
レイが相手メッシュと交差しています。
距離：8.8
  平面方程式 0.0x+ 0.0y+ -1.0z+-6.2=0
```

●RayIntersect関数内

```
//レイが交差しているメッシュ内ポリゴンの3頂点を取得する
D3DXVECTOR3 vVertex[3];
FindVerticesOnPoly(pThingB->pMesh,dwIndex,vVertex);
//その３頂点を基に、平面方程式を計算する
D3DXPlaneFromPoints(pPlane,&vVertex[0],&vVertex[1],&vVertex[2]);
```

●FindVerticesOnPoly関数

```
//そのポリゴンの頂点を見つける
HRESULT FindVerticesOnPoly(LPD3DXMESH pMesh,DWORD dwPolyIndex,D3DXVECTOR3＊
pvecVertices )
{
    DWORD i,k;
    DWORD dwStride=pMesh->GetNumBytesPerVertex();
    DWORD dwVertexAmt=pMesh->GetNumVertices();
    DWORD dwPolyAmt=pMesh->GetNumFaces();
    WORD＊ pwPoly=NULL;
    pMesh->LockIndexBuffer(D3DLOCK_READONLY,(VOID＊＊)&pwPoly);

    BYTE ＊pbVertices=NULL;
    FLOAT＊ pfVetices=NULL;
    LPDIRECT3DVERTEXBUFFER9 VB=NULL;
    pMesh->GetVertexBuffer(&VB);
    if(SUCCEEDED(VB->Lock(0,0,(VOID＊＊)&pbVertices,0)))
    {
            pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex＊ 3]
];
            pvecVertices[0].x=pfVetices[0];
            pvecVertices[0].y=pfVetices[1];
```

```
                pvecVertices[0].z=pfVetices[2];

                pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
        3+1] ];
                pvecVertices[1].x=pfVetices[0];
                pvecVertices[1].y=pfVetices[1];
                pvecVertices[1].z=pfVetices[2];

                pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
        3+2] ];
                pvecVertices[2].x=pfVetices[0];
                pvecVertices[2].y=pfVetices[1];
                pvecVertices[2].z=pfVetices[2];

                pMesh->UnlockIndexBuffer();
                VB->Unlock();
            }
            return S_OK;
        }
```

235 線(レイ)とメッシュ(内ポリゴン)の交点座標を求める

●D3DXIntersect関数から得た重心座標をもとに交点座標を算出する　中級

サンプルプロジェクト　235　レイとメッシュの交点座標

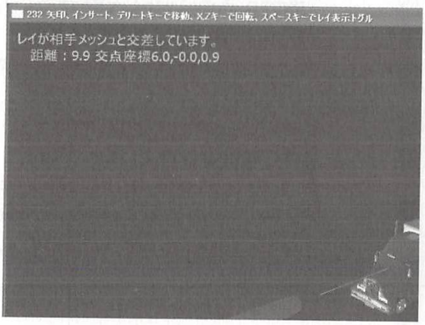
D3DXIntersect関数は一見すると交点座標を返さないように見えますが、第6、第7引数に格納される重心座標から交点は簡単に求まります。

重心座標をU,V、ポリゴンの3頂点をP0,P1,P2とすると、交点Pは次の式で求まります。

P=P0+U＊(P1−P0)+V＊(P2−P0)

サンプルのコードでは、この式をそのまま交点の算出に使用しています。

▼実行結果


232 矢印、インサート、デリートキーで移動、XZキーで回転、スペースキーでレイ表示トグル

```
レイが相手メッシュと交差しています。
  距離：9.9 交点座標6.0,-0.0,0.9
```

```
DWORD dwIndex=0;
D3DXVECTOR3 vVertex[3];
FLOAT U=0,V=0;
vecDirection=vecEnd-vecStart;
D3DXIntersect(pThingB->pMesh,&vecStart,&vecDirection,&boHit,&dwIndex,&U,&V,
pfDistance,NULL,NULL);
if(boHit)
{
    FindVerticesOnPoly(pThingB->pMesh,dwIndex,vVertex);
    //重心座標から交点を算出　ローカル交点pは　v0+U*(v1-v0)+V*(v2-v0)
    *pvIntersect=vVertex[0]+U*(vVertex[1]-vVertex[0])+V*(vVertex[2]-
vVertex[0]);
    //ローカル交点をワールド系に換算
    D3DXVec3TransformCoord(pvIntersect,pvIntersect,&pThingB->matWorld);
    return TRUE;
}
```

236 直線（レイ）とアニメーションメッシュの交差を検出する

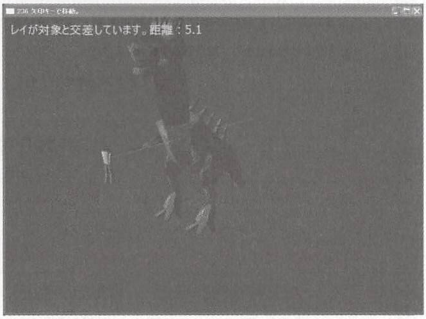
D3DXIntersect関数を利用してアニメーションメッシュとの交差判定を行う　上級

サンプルプロジェクト　236 レイとアニメーションメッシュ

アニメーションメッシュは、単一のメッシュではなく、いくつかの“パーツメッシュ”を階層構造で連結し、それぞれの相対姿勢を変化させるものです。アニメーションメッシュにレイを当てるとはそれらのパーツメッシュのどれかに当てることとも言えます。

アニメーションメッシュのレイ判定と言っても、若干の違いはあるものの、個々のパーツメッシュごとの判定部分はTips231と同様です。アニメーションメッシュ固有の処理としては、階層の走査が伴うことぐらいです。

▼実行結果



レイが対象と交差しています。距離：5.1

```
BOOL Collision(LPD3DXFRAME pFrameBase,THING＊ pThingA ,THING＊ pThingB,BOOL
＊ pboHit,
                FLOAT＊ pfDistance,LPD3DXMESHCONTAINER＊ ppContainer)
{
    MYFRAME ＊pFrame = (MYFRAME＊)pFrameBase;
    if(NULL !=pFrame->pMeshContainer)
    {
          LPD3DXMESH pMesh=pFrame->pMeshContainer->MeshData.pMesh;
          if(NULL != pMesh)
          {
                D3DXMATRIX matWorld,matRotation;
                D3DXVECTOR3 vecStart,vecEnd,vecDirection;
                vecStart=pThingA->vecPosition;
                vecEnd=D3DXVECTOR3(0,0,1);
                D3DXMatrixRotationY(&matRotation,pThingA->fYaw);
                D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matRotation);
                vecEnd+=vecStart;
                //　対象が動いている場合でも対象のワールド行列の逆行列を用いれば正しくレイ
が当たる
                D3DXMatrixTranslation(&matWorld,pThingB->vecPosition.x,
pThingB->vecPosition.y,pThingB->vecPosition.z);
                D3DXMatrixMultiply(&matWorld,&pFrame-
>CombinedTransformationMatrix,&matWorld);
                D3DXMatrixInverse(&matWorld,NULL,&matWorld);
                D3DXVec3TransformCoord(&vecStart,&vecStart,&matWorld);
                D3DXVec3TransformCoord(&vecEnd,&vecEnd,&matWorld);
                vecDirection=vecEnd-vecStart;

D3DXIntersect(pMesh,&vecStart,&vecDirection,pboHit,NULL,NULL,NULL,
pfDistance,NULL,NULL);
                if(＊pboHit)
                {
                      ＊ppContainer=pFrame->pMeshContainer;
                      return TRUE;
                }
          }
    }
    //　再帰判断部分　（階層の走査）
    if(pFrame->pFrameSibling != NULL)
    {
          if(Collision(pFrame->pFrameSibling,pThingA,pThingB,pboHit,
pfDistance,ppContainer)==TRUE)
          {
                return TRUE;
```

```
           }
    }
    if(pFrame->pFrameFirstChild != NULL)
    {
           if(Collision(pFrame->pFrameFirstChild,pThingA,pThingB,pboHit,
pfDistance,ppContainer)==TRUE)
           {
                 return TRUE;
           }
    }
    return FALSE;
}
```

237 交差しているアニメーションメッシュ内のパーツメッシュを得る

●メッシュコンテナーに格納されている情報からパーツメッシュの名前を得る　上級

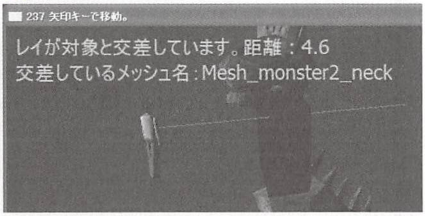
サンプルプロジェクト　237　交差しているパーツメッシュ

直前のサンプルと本質的な部分ではまったく同じであり、99%同じコードです。

実は直前サンプルのコードでも、パーツメッシュの名前を取り出すことはすぐに出来ることであり、ただそれをしていなかっただけです。

それぞれのパーツメッシュを再帰的に走査し、レイが当たってるのメッシュコンテナーのNameメンバ変数（あるいはパーツフレームのNameメンバ変数でもいい）を参照すれば難なく名前を知ることが出来ます。

▼実行結果



237 矢印キーで移動。

```
レイが対象と交差しています。距離：4.6
交差しているメッシュ名：Mesh_monster2_neck
```

●Main.cpp側

```
Collision(Thing[1].pFrameRoot,&Thing[0],&Thing[1],&boHit,&fDistance,&pContai
ner);
 if(boHit)
 {
     TCHAR szStr[MAX_PATH+1];
     sprintf(szStr,"レイが対象と交差しています。距離 : %3.1f", fDistance);
```

```
        RenderString(szStr,10,10);
        sprintf(szStr,"交差しているメッシュ名：%s",pContainer->Name);
        RenderString(szStr,10,40);
    }
```

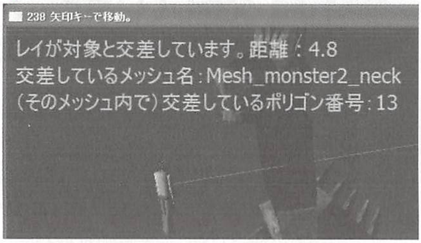
238 交差しているアニメーションメッシュ内のポリゴンを得る

●アニメーションメッシュ内部でレイが当たっているポリゴンを特定する　上級

サンプルプロジェクト　238　パーツメッシュ内ポリゴン

Tips233と手順は同様です。相違はアニメーションゆえ、パーツメッシュ（パーツフレーム）の走査が必要なことぐらいです。

▼実行結果



238 矢印キーで移動。

```
レイが対象と交差しています。距離：4.8
交差しているメッシュ名:Mesh_monster2_neck
(そのメッシュ内で)交差しているポリゴン番号:13
```

```
BOOL boHit=FALSE;
 FLOAT fDistance=0;
 DWORD dwIndex=0;
 LPD3DXMESHCONTAINER pContainer = NULL;
 Collision(Thing[1].pFrameRoot,&Thing[0],&Thing[1],&boHit,&fDist
0ance,&pContainer,&dwIndex);
 if(boHit)
 {
     TCHAR szStr[MAX_PATH+1];
     sprintf(szStr,"レイが対象と交差しています。距離 : %3.1f", fDistance);

     RenderString(szStr,10,10);
     sprintf(szStr,"交差しているメッシュ名：%s",pContainer->Name);
     RenderString(szStr,10,40);
     sprintf(szStr,"（そのメッシュ内で）交差しているポリゴン番号：%d",dwIndex);

     RenderString(szStr,10,70);
 }
```

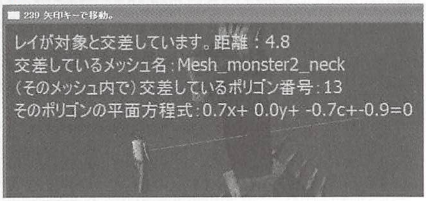
239 交差しているアニメーションメッシュ内ポリゴンの平面方程式を計算する

●アニメーションのパーツメッシュ内ポリゴンを構成する3つの頂点を抽出して平面方程式を導く 初級

サンプルプロジェクト 239 Animポリゴンの平面方程式

Tips234と手順は同様です。相違はアニメーションゆえ、パーツメッシュ（パーツフレーム）の走査が必要なことぐらいです。

▼実行結果



```
239 矢印キーで移動。
レイが対象と交差しています。距離：4.8
交差しているメッシュ名：Mesh_monster2_neck
（そのメッシュ内で）交差しているポリゴン番号：13
そのポリゴンの平面方程式：0.7x+ 0.0y+ -0.7c+-0.9=0
```

●RayIntersect.cpp側

```
BOOL Collision(LPD3DXFRAME pFrameBase,THING＊ pThingA ,THING＊ pThingB,BOOL
 ＊ pboHit,
                    FLOAT＊ pfDistance,LPD3DXMESHCONTAINER＊ ppContainer,DWORD＊
pdwIndex,D3DXPLANE＊ pP)
D3DXIntersect(pMesh,&vecStart,&vecDirection,pboHit,pdwIndex,NULL,NULL,
pfDistance,NULL,NULL);
if(＊pboHit)
{
    ＊ppContainer=pFrame->pMeshContainer;
    D3DXVECTOR3 v[3];
    FindVerticesOnPoly(pMesh,＊pdwIndex,v);
    D3DXPlaneFromPoints(pP,&v[0],&v[1],&v[2]);
    return TRUE;
}
```

●Main.cpp側

```
BOOL boHit=FALSE;
 FLOAT fDistance=0;
 DWORD dwIndex=0;
 D3DXPLANE P;
 LPD3DXMESHCONTAINER pContainer = NULL;
 Collision(Thing[1].pFrameRoot,&Thing[0],&Thing[1],&boHit,
 &fDistance,&pContainer,&dwIndex,&P);
 if(boHit)
```

```
    {
        TCHAR szStr[MAX_PATH+1];
        sprintf(szStr,"レイが対象と交差しています。距離 : %3.1f", fDistance);

        RenderString(szStr,10,10);
        sprintf(szStr,"交差しているメッシュ名：%s",pContainer->Name);
        RenderString(szStr,10,40);
        sprintf(szStr,"（そのメッシュ内で）交差しているポリゴン番号：%d",dwIndex);

        RenderString(szStr,10,70);
        sprintf(szStr,"そのポリゴンの平面方程式：%3.1fx+ %3.1fy+ %3.1fc+%3.1f=0"
                ,P.a,P.b,P.c,P.d);
        RenderString(szStr,10,100);
    }
```

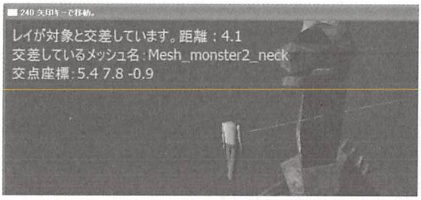
240 線（レイ）とアニメーションメッシュ（内ポリゴン）の交点座標を求める

●D3DXIntersect関数から得た重心座標をもとに交点座標を算出する　上級

サンプルプロジェクト　240 Ray-Anim交点座標

Tips235と手順は同様です。相違はアニメーションゆえ、パーツメッシュ（パーツフレーム）の走査が必要なことぐらいです。

▼実行結果



レイが対象と交差しています。距離：4.1
交差しているメッシュ名：Mesh_monster2_neck
交点座標：5.4 7.8 -0.9

●RayIntersect.cpp側

```
if(＊pboHit)
{
     ＊ppContainer=pFrame->pMeshContainer;
    FindVerticesOnPoly(pMesh,dwIndex,vVertex);
     //重心座標から交点を算出　ローカル交点pは　v0+U＊(v1-v0)+V＊(v2-v0)
     ＊pvIntersect=vVertex[0]+U＊(vVertex[1]-vVertex[0])+V＊(vVertex[2]-
vVertex[0]);
    //ローカル交点をワールド系に換算 特にアニメーションの場合、これをしないと
```

```
   //モーションしているにも関らず（つまりワールド系で位置が変わっても）同じ座標になってしま
う
   D3DXVec3TransformCoord(pvIntersect,pvIntersect,&pFrame-
>CombinedTransformationMatrix);

   return TRUE;
}

Main.cpp側
BOOL boHit=FALSE;
 FLOAT fDistance=0;
 LPD3DXMESHCONTAINER pContainer = NULL;
 D3DXVECTOR3 vIntersect;

Collision(Thing[1].pFrameRoot,&Thing[0],&Thing[1],&boHit,&fDistance,&pContai
ner,&vIntersect);
 if(boHit)
 {
     TCHAR szStr[MAX_PATH+1];
     sprintf(szStr,"レイが対象と交差しています。距離 : %3.1f", fDistance);

     RenderString(szStr,10,10);
     sprintf(szStr,"交差しているメッシュ名：%s",pContainer->Name);
     RenderString(szStr,10,40);
     sprintf(szStr,"交点座標：%3.1f %3.1f %3.1f",vIntersect.x,vIntersect.y,
vIntersect.z);
     RenderString(szStr,10,70);
 }
```

241 バウンディングスフィアとは？

バウンディングスフィアの意味を理解する

初級

バウンディングスフィアを日本語で言うと境界球です。言葉通り球状の領域のことです。

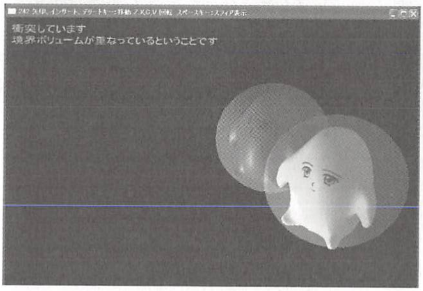
これだけではどのような局面で使用されるのかが分かりませんね。 バウンディングスフィアは衝突判定、いわゆる「当たり判定」に使用します。物体と物体が（本書ではメッシュとメッシュが）離れているのか、接触しているのかを判断する一つの手法です。

バウンディングスフィアは、半径と2つの物体の距離の「大小関係」だけで衝突を検出することができるため、最も高速な判定が出来ます。以降で解説する軸平行バウンディン

グボックスも、大小関係で判断できる高速な判定ではありますが、バウンディングスフィアほどではありません。

ただし、バウンディングスフィアは非常に高速な手法なのですが、やはりそこには“処理速度と判定精度”のトレードオフがあります。境界球は最も高速であると同時に、最も「大雑把」な判定でもあることに留意しましょう。

▼実行結果


242 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：スフィア表示

衝突しています
境界ボリュームが重なっているということです

242 バウンディングスフィアにより衝突判定する

D3DXComputeBoundingSphere関数によりバウンディングスフィアを得る　初級

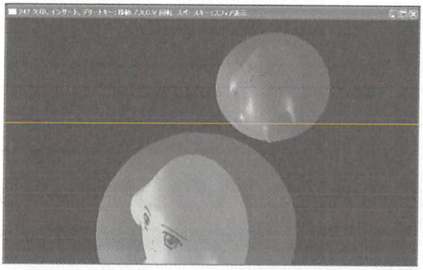
サンプルプロジェクト　242 Bスフィア

バウンディングスフィアを作成するためにD3DXComputeBoundingSphere関数が用意されています。この関数は　スフィア作成に必要な半径と中心を計算するためにメッシュの頂点にアクセスします。そのためこの関数を実行する前に頂点バッファをロックしなくてはなりません。

引数は次のとおりです。

第1引数、Direct3Dデバイスのポインタ
第2引数、頂点数
第3引数、頂点フォーマットタイプ
第4引数、スフィアの中心を格納したいベクトルのアドレス
第5引数、スフィアの半径を格納したいFLOAT変数のアドレス

▼実行結果


242 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：スフィア表示

●スフィアの作成

```
//
//HRESULT InitSphere(LPDIRECT3DDEVICE9 pDevice,THING* pThing)
//スフィアの計算およびスフィアを視認可能にするためにスフィアメッシュを作成する
HRESULT InitSphere(LPDIRECT3DDEVICE9 pDevice,THING* pThing)
{
    HRESULT hr = NULL;
    LPDIRECT3DVERTEXBUFFER9 pVB = NULL;
    VOID＊ pVertices = NULL;
    D3DXVECTOR3 vecCenter;
    FLOAT fRadius;

    //メッシュの頂点バッファーをロックする
    if(FAILED(pThing->pMesh->GetVertexBuffer( &pVB )))
    {
          return E_FAIL;
    }
    if(FAILED(pVB->Lock( 0, 0, &pVertices, 0 )))
    {
        SAFE_RELEASE( pVB );
        return E_FAIL;
    }
    // メッシュの外接円の中心と半径を計算する
    hr=D3DXComputeBoundingSphere( (D3DXVECTOR3＊)pVertices, pThing->pMesh-
>GetNumVertices(),
                                       D3DXGetFVFVertexSize(pThing->pMesh-
>GetFVF()), &vecCenter,
                                     &fRadius );
    pVB->Unlock();
    SAFE_RELEASE( pVB );

    if(FAILED( hr ))
    {
        return hr;
    }
    pThing->Sphere.vecCenter=vecCenter;
    pThing->Sphere.fRadius=fRadius;
```

●分かりやすいようにスフィアを可視化するための処理部分（本質ではありません）

```
    //これ以降の行は本質ではない。確認のためスフィア自体をレンダリングするための準備に過ぎない

    //  得られた中心と半径を基にメッシュとしてのスフィアを作成する
    hr=D3DXCreateSphere(pDevice,fRadius,24,24,&pThing->pSphereMesh,NULL);
    if(FAILED(  hr  ))
    {
          return  hr;
    }
    //スフィアメッシュのマテリアル　白色、半透明、光沢強
    pThing->pSphereMeshMaterials  =  new  D3DMATERIAL9;
    pThing->pSphereMeshMaterials->Diffuse.r=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.g=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.b=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.a=0.3f;
    pThing->pSphereMeshMaterials->Ambient=pThing->pSphereMeshMaterials-
>Diffuse;
    pThing->pSphereMeshMaterials->Specular.r=1.0f;
    pThing->pSphereMeshMaterials->Specular.g=1.0f;
    pThing->pSphereMeshMaterials->Specular.b=1.0f;
    pThing->pSphereMeshMaterials->Emissive.r=0.1f;
    pThing->pSphereMeshMaterials->Emissive.g=0.1f;
    pThing->pSphereMeshMaterials->Emissive.b=0.1f;
    pThing->pSphereMeshMaterials->Power=120.0f;

    return  S_OK;
}
```

●スフィアによる衝突判断

```
//
//BOOL Collision(THING＊ pThingA,THING＊ pThingB)
//衝突検出
BOOL Collision(THING＊ pThingA,THING＊ pThingB)
{
    //2つの物体の中心間の距離を求める
    D3DXVECTOR3 vecLength=pThingB->vecPosition-pThingA->vecPosition;
    FLOAT fLength=D3DXVec3Length(&vecLength);
    // その距離が、2物体の半径を足したものより小さいということは、
    //境界球同士が重なっている（衝突している）ということ
    if(fLength <= pThingA->Sphere.fRadius+pThingB->Sphere.fRadius)
    {
        return TRUE;
    }
    return FALSE;
}
```

2つのメッシュの距離（2つのメッシュの中心同士の距離）をL、それぞれのメッシュの　バウンディングスフィアの半径をR1、R2とすると、
L<= (R1+R2)のとき、衝突
L> (R1+R2)のとき、非衝突
と簡単に判断できます。

コード上では、これと全く同じ判断をしています。

if(fLength <= pThingA->Sphere.fRadius+pThingB->Sphere.fRadius)

fLengthは2つのメッシュ間の距離(L)、pThingA->Sphere.fRadiusはメッシュAの半径、pThingB->Sphere.fRadiusはメッシュBの半径です。

243 バウンディングスフィアにより衝突判定する(アニメーションメッシュ)

●D3DXFrameCalculateBoundingSphere関数により フレーム全体としての バウンディングスフィアを作成する

初級

サンプルプロジェクト 243 BスフィアAnim

アニメーションメッシュの場合はD3DXFrameCalculateBoundingSphere関数を使用すれば、全パーツメッシュを包み込む全体としてのバウンディングスフィアを得ることが出来ます。

第1引数にはアニメーションのルートフレームのポインタを渡します。第2、第3引数に渡したアドレスに、それぞれ中心位置ベクトルと半径を格納してくれます。単なるメッシュではないので、フレームを渡すことがポイントです。フレームから色々な情報を取り出せるので、逆にD3DXComputeBoundingSphereよりアプリ側から渡す引数はシンプルです。

```
HRESULT InitSphere(LPDIRECT3DDEVICE9 pDevice,THING* pThing)
{
    // メッシュの外接円の中心と半径を計算する
    D3DXFrameCalculateBoundingSphere(pThing->pFrameRoot,&pThing-
>Sphere.vecCenter,
            &pThing->Sphere.fRadius);
```

●分かりやすいようにスフィアを可視化するための処理部分（本質ではありません）

```
    // 得られた中心と半径を基にメッシュとしてのスフィアを作成する
    D3DXCreateSphere(pDevice,pThing->Sphere.fRadius,24,24,&pThing-
>pSphereMesh,NULL);
    //スフィアメッシュのマテリアル　白色、半透明、光沢強
    pThing->pSphereMeshMaterials = new D3DMATERIAL9;
    pThing->pSphereMeshMaterials->Diffuse.r=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.g=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.b=1.0f;
    pThing->pSphereMeshMaterials->Diffuse.a=0.3f;
    pThing->pSphereMeshMaterials->Ambient=pThing->pSphereMeshMaterials-
>Diffuse;
    pThing->pSphereMeshMaterials->Specular.r=1.0f;
    pThing->pSphereMeshMaterials->Specular.g=1.0f;
    pThing->pSphereMeshMaterials->Specular.b=1.0f;
    pThing->pSphereMeshMaterials->Emissive.r=0.1f;
    pThing->pSphereMeshMaterials->Emissive.g=0.1f;
    pThing->pSphereMeshMaterials->Emissive.b=0.1f;
    pThing->pSphereMeshMaterials->Power=120.0f;

    return S_OK;
}
```

●衝突判定部分

```
BOOL Collision(THING＊ pThingA,THING＊ pThingB)
{
    //２つの物体の中心間の距離を求める
    D3DXVECTOR3 vecLength=pThingB->vecPosition-pThingA->vecPosition;
    FLOAT fLength=D3DXVec3Length(&vecLength);
    // その距離が、2物体の半径を足したものより小さいということは、
    //境界球同士が重なっている（衝突している）ということ
    if(fLength <= pThingA->Sphere.fRadius+pThingB->Sphere.fRadius)
    {
            return TRUE;
    }
    return FALSE;
}
```

判定部分の原理は、直前のTipsと同じで、2つのメッシュの距離と半径の合計を比較しているだけです。

●注意しなければならない点

アニメーションメッシュは、当然動きがありますから、全体としての バウンディングスフィアのサイズは常に変化します。

したがって、メッシュの位置を移動しなくても、モーションの影響でスフィアのサイズが変化して、衝突したり、しなかったりします。（逆に、そうなるように実装しなくてはなりません）

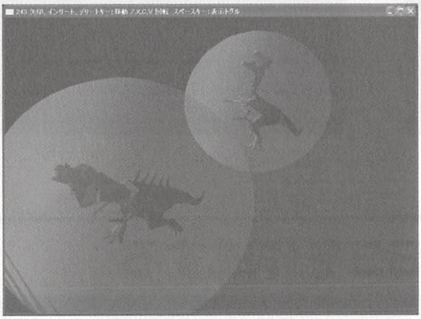
つまり、スフィアのサイズを常に再計算しなくてはならないということです。その時点でのモーションに対するスフィアを動的に作成していきます。

次の2つの図は、メッシュの位置はまったく同じですが、モーションによってスフィアのサイズが変化して、衝突したりしなかったりしています。

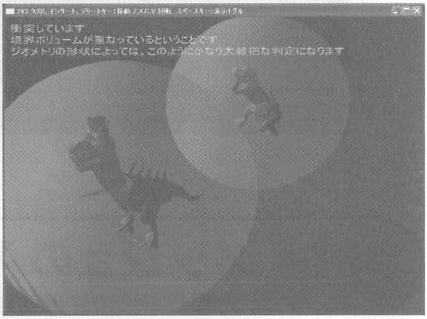
本サンプルでこのスフィアの動的更新を行っているのは、UpdateSphere関数です。

```
HRESULT UpdateSphere(LPDIRECT3DDEVICE9 pDevice,THING＊ pThing)
{
    D3DXFrameCalculateBoundingSphere(pThing->pFrameRoot,&pThing-
>Sphere.vecCenter,
            &pThing->Sphere.fRadius);
    pThing->pSphereMesh->Release();
    D3DXCreateSphere(pDevice,pThing->Sphere.fRadius,24,24,&pThing-
>pSphereMesh,NULL);

        return S_OK;
}
```

▼実行結果


243 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

▼実行結果


243 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

衝突しています
境界ボリュームが重なっているということです
ジオメトリの形状によっては、このようにかなり大雑把な判定になります

基底軸平行バウンディングボックスAABB

244 AABBとは？

AABBの意味を理解する

初級

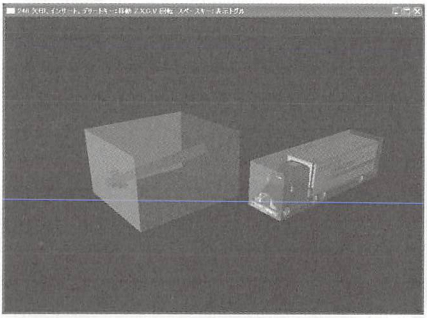
AABBはAxis Aligned BoudingBox（アクシス・アラインド・バウンディングボックス）の略語で、日本語で言うと軸平行バウンディングボックスとなります。つまり立方体でかつ基底軸（x,y,z軸）に平行つまり回転しない（回転させてはいけない）境界ボリュームのことです。

AABBも境界球のように、「大小関係」で判断します。具体的には、物体同士の頂点全ての大小を比較することにより、一方が他方の物体に接触しているか、重なっているか、及び含んでいるかということを判断します。

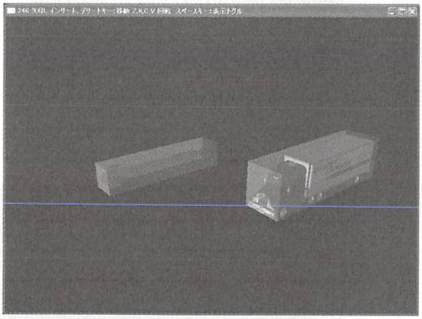
AABBには難点があります。それは「メッシュは通常回転する」ということです。メッシュが回転しているのにボックスが回転しない場合どういうことになるかは予想ができるでしょう。かと言ってメッシュと同期してボックスを回転させてしまってはAABBとは言えなくなりますし、当然AABBゆえのメリットを失います。

そこで軸平行を維持するために、回転の度に最大頂点と最小頂点を再計算します。つまり、バウンディングボックスを回転の度に動的に作り直します。これにより姿勢によってボックスが大きくなったり小さくなったりしますが、軸平行を維持するためには仕方のないことです。

▼実行結果


246 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

▼実行結果


246 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

245 メッシュのAABBを作成する

●AABBは動的に再構築していかなくてはならない

中級

参考サンプルプロジェクト　246 AABB

直前のTipsで述べたとおり、AABBを維持するためにはメッシュの回転に伴ってボックスを再構築しなくてはなりません。

具体的な流れは次のとおりです。

- メッシュの頂点バッファをロックする。
- 全頂点に対し、メッシュの回転を掛ける（頂点を座標変換する）
- 回転を掛けた後の全メッシュのなかから最大、最小を見つける。言い換えると　バウンディングボックスを作る。

サンプルでは、この処理を行うためにUpdateBBoxMesh関数を作成しています。

```
HRESULT UpdateBBoxMesh(LPDIRECT3DDEVICE9 pDevice,THING* pThing)
{
    HRESULT hr = NULL;
    LPD3DXMESH pClonedMesh=NULL;
    LPDIRECT3DVERTEXBUFFER9 pVB=NULL;
    BYTE* pVertices = NULL;
    DWORD dwVertexAmount=0;
    DWORD dwStride=0;
    D3DXVECTOR3 vecMax,vecMin;

    dwVertexAmount=pThing->pBBoxMeshHold->GetNumVertices();
    dwStride=D3DXGetFVFVertexSize(pThing->pBBoxMeshHold->GetFVF());
    pThing->pBBoxMeshHold->CloneMeshFVF( D3DXMESH_MANAGED,pThing-
>pBBoxMeshHold->GetFVF(),pDevice,&pClonedMesh);

        //ボックスメッシュの頂点バッファーをロックする
        if(FAILED(pClonedMesh->GetVertexBuffer( &pVB )))
        {
                return E_FAIL;
        }
        if(FAILED(pVB->Lock( 0, 0, (VOID＊＊)&pVertices, 0 )))
        {
            SAFE_RELEASE( pVB );
            return E_FAIL;
        }
        //ボックスメッシュの頂点に回転をかける
        for(DWORD i=0;i<dwVertexAmount ＊ dwStride;i+=dwStride)
        {
                D3DXVec3TransformCoord( (D3DXVECTOR3＊)&pVertices[i],(D3DXVECTOR3
＊)&pVertices[i],&pThing->matRotation);
        }
        // ボックスメッシュの内頂点位置の最大と最小を検索する
        if(FAILED(D3DXComputeBoundingBox( (D3DXVECTOR3＊)pVertices,
dwVertexAmount,
                                              dwStride, &vecMin,&vecMax )))
        {
            return hr;
        }
        //もう一時的な頂点バッファー領域に用はない。アンロックしてリリース
        pVB->Unlock();
        SAFE_RELEASE( pVB );
        //BBOX情報を保存
        pThing->BBox.vecMax=vecMax;
        pThing->BBox.vecMin=vecMin;

        //現在のボックスメッシュを破棄
        SAFE_RELEASE( pClonedMesh );
        SAFE_RELEASE( pThing->pBBoxMesh );
        // 新たなを最大位置と最小位置を基にボックスメッシュを新規作成する
        hr=D3DXCreateBox(pDevice,vecMax.x＊2,vecMax.y＊2,vecMax.z＊2,&pThing-
>pBBoxMesh,NULL);
        if(FAILED( hr ))
        {
            return hr;
        }
        return S_OK;
}
```

246 AABBにより衝突判定する

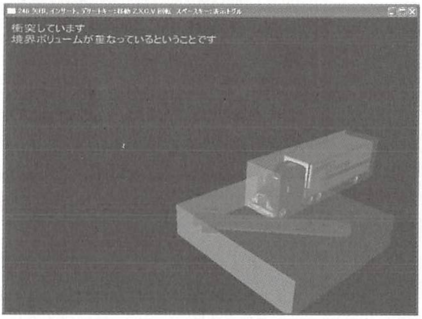
●AABBなので頂点の大小関係だけで判定できる

中級

サンプルプロジェクト　246 AABB

衝突判定部分であるAABBCollisionDetection関数の中身を見てもらえば分かるとおり、単なる大小関係しか判断していません。しかも、本質的な判定部分はif文がたった1行です。

▼実行結果


246 矢印、インサート、デリートキー：移動 Z,X,C,V 回転　スペースキー：表示トグル

衝突しています
境界ボリュームが重なっているということです

```
if(vecMinA.x < vecMaxB.x && vecMaxA.x > vecMinB.x && vecMinA.y < vecMaxB.y
            && vecMaxA.y > vecMinB.y && vecMinA.z < vecMaxB.z && vecMaxA.z >
vecMinB.z)
```

AABBの場合、判定そのものは高速に行えますが、実はAABBを維持する処理のほうが時間がかかります。

●衝突判定部分

```
BOOL AABBCollisionDetection(THING* pThingA,THING* pThingB)
{
    //2つの物体をA,Bとして、それぞれの最大、最小位置を求める
    D3DXVECTOR3 vecMaxA,vecMinA,vecMaxB,vecMinB;
    vecMaxA=pThingA->BBox.vecMax+pThingA->vecPosition;
    vecMinA=pThingA->BBox.vecMin+pThingA->vecPosition;
    vecMaxB=pThingB->BBox.vecMax+pThingB->vecPosition;
    vecMinB=pThingB->BBox.vecMin+pThingB->vecPosition;
    //2つの物体それぞれの最大、最小位置という4つの情報があれば衝突を検出できる
    if(vecMinA.x < vecMaxB.x && vecMaxA.x > vecMinB.x && vecMinA.y <
vecMaxB.y
            && vecMaxA.y > vecMinB.y && vecMinA.z < vecMaxB.z && vecMaxA.z >
vecMinB.z)
    {
            return TRUE;
    }
    return FALSE;
}
```

有向バウンディングボックスOBB

247 OBBとは？

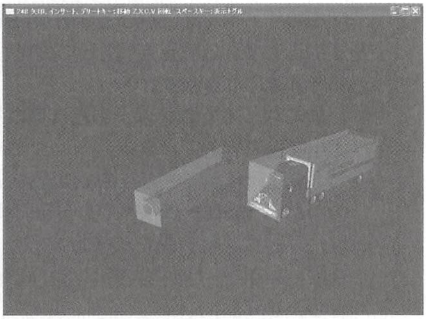
●OBBの意味を理解する

初級

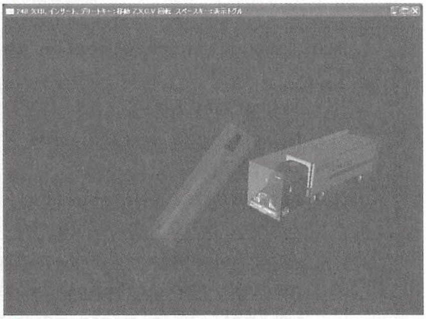
OBBはOriented Bouding Box（オリエンティッド・バウンディングボックス）の略語で、日本語で言うと有向バウンディングボックスとなります。OBBはボックスに向きを持たせることができます。つまりボックスをメッシュに同期して回転させることができます。AABBの場合、ボックスを回転させることが出来ないため、少々無理やりな準備が必要になりコーディングがかえって面倒だったり、それ以上になんといっても判定精度が低かった（スフィアよりはマシですが）のに対し、このOBBはその欠点を全て解消します。

図を見ると明らかなように、メッシュが回転していてもボックスの形状を変化させる必要が無いので、これまでのボリューム（スフィア、AABB）よりもデッドスペースを最小限にすることができます。

▼実行結果


248 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

▼実行結果


248 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

248 OBBにより衝突判定する

●“15本の分離軸”により回転しているボックスでも判定を可能にする　中級

サンプルプロジェクト　248 OBB

OBBにおける判定原理を簡潔に述べると、ボックスの“影”に着目して、その影同士の交差の有無で判定しようというものです。

“影”が離れていればボックスが離れている、つまり衝突していないということになります。

ボックスの影とは、3次元のボックスを1次元に投影したものを意味します。影を求めるということは、投影することです。読んで字の如く“影”を“投げる”のが投影あるいは射影です。3次元を1次元に投影したときの影は線になります。

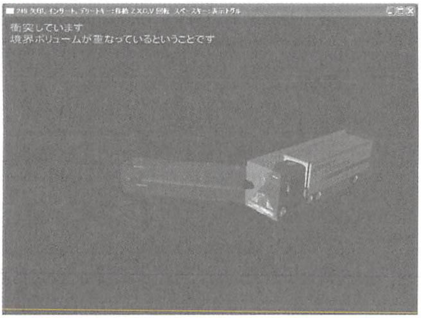
線になってくれれば、あとは楽なもので、大小関係で判定できます。

1次元と言ってもどこの1次元に投影するのかという疑問が浮かぶでしょう。具体的には、ボックスAとボックスBがあったとして、まず、AのローカルなX軸、Y軸、Z軸上に投影します（3本）。

次に、同じくBのローカルなX軸、Y軸、Z軸上に投影します（3本）。そして、AとBのローカル軸ベクトル同士の外積ベクトル上に投影します（3と3の組み合わせ=9本）。3本+3本+9本=15本の線上に投影します。そして、15本のうち一本でも影が離れていれば、衝突していないという判断が出来ます。

サンプルではこの原理をそのままコードにしています。異なる部分は一つもありません。

▼実行結果


248 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

衝突しています
境界ボリュームが重なっているということです

```
BOOL OBBCollisionDetection(THING＊ pThingA,THING＊ pThingB)
{
    //ボックスの距離ベクトル（中心と中心を結んだベクトル）
    D3DXVECTOR3 vecDistance = pThingB->vecPosition - pThingA->vecPosition;
    //分離軸
    D3DXVECTOR3 vecSeparate;
    //それぞれのローカル基底軸ベクトルに、それぞれの回転を反映させる
    pThingA->BBox.vecAxisX=D3DXVECTOR3(1,0,0);
    pThingA->BBox.vecAxisY=D3DXVECTOR3(0,1,0);
    pThingA->BBox.vecAxisZ=D3DXVECTOR3(0,0,1);

    pThingB->BBox.vecAxisX=D3DXVECTOR3(1,0,0);
    pThingB->BBox.vecAxisY=D3DXVECTOR3(0,1,0);
    pThingB->BBox.vecAxisZ=D3DXVECTOR3(0,0,1);

    D3DXVec3TransformCoord(&pThingA->BBox.vecAxisX,&pThingA-
>BBox.vecAxisX,&pThingA->matRotation);
    D3DXVec3TransformCoord(&pThingA->BBox.vecAxisY,&pThingA-
>BBox.vecAxisY,&pThingA->matRotation);
    D3DXVec3TransformCoord(&pThingA->BBox.vecAxisZ,&pThingA-
>BBox.vecAxisZ,&pThingA->matRotation);

    D3DXVec3TransformCoord(&pThingB->BBox.vecAxisX,&pThingB-
>BBox.vecAxisX,&pThingB->matRotation);
    D3DXVec3TransformCoord(&pThingB->BBox.vecAxisY,&pThingB-
>BBox.vecAxisY,&pThingB->matRotation);
    D3DXVec3TransformCoord(&pThingB->BBox.vecAxisZ,&pThingB-
>BBox.vecAxisZ,&pThingB->matRotation);
    //ボックスAのローカル基底軸を基準にした、”影”の算出（3パターン）
    {
        //X軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingA-
>BBox.vecAxisX,&vecDistance))return FALSE;
        //Y軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingA-
>BBox.vecAxisY,&vecDistance))return FALSE;
        //Z軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingA-
>BBox.vecAxisZ,&vecDistance))return FALSE;
    }
    //ボックスBのローカル基底軸を基準にした、”影”の算出（3パターン）
    {
        //X軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingB-
>BBox.vecAxisX,&vecDistance))return FALSE;
        //Y軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingB-
>BBox.vecAxisY,&vecDistance))return FALSE;
        //Z軸を分離軸とした場合
        if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&pThingB-
>BBox.vecAxisZ,&vecDistance))return FALSE;
    }
    //お互いの基底軸同士の外積ベクトルを基準にした、”影”の算出（9パターン）
    {
        //ボックスAのX軸
        {
            //と　ボックスBのX軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisX,&pThingB->BBox.vecAxisX);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのY軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisX,&pThingB->BBox.vecAxisY);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのZ軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisX,&pThingB->BBox.vecAxisZ);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
        }
        //ボックスAのY軸
        {
            //と　ボックスBのX軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisY,&pThingB->BBox.vecAxisX);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのY軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisY,&pThingB->BBox.vecAxisY);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのZ軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisY,&pThingB->BBox.vecAxisZ);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
        }
        //ボックスAのZ軸
        {
            //と　ボックスBのX軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisZ,&pThingB->BBox.vecAxisX);
            if(!CompareLength(&pThingA->BBox,&pThingB->BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのY軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA->BBox.vecAxisZ,&pThingB->BBox.vecAxisY);
            if(!CompareLength(&pThingA->BBox,&pThingB-
>BBox,&vecSeparate,&vecDistance)) return FALSE;
            //と　ボックスBのZ軸との外積ベクトルを分離軸とした場合
            D3DXVec3Cross( &vecSeparate, &pThingA-
>BBox.vecAxisZ,&pThingB->BBox.vecAxisZ);
            if(!CompareLength(&pThingA->BBox,&pThingB-
>BBox,&vecSeparate,&vecDistance)) return FALSE;
        }
    }
    return TRUE;
}
```

```
//
//BOOL CompareLength(BBOX＊ pboxA,BBOX＊ pboxB,D3DXVECTOR3＊
pvecSeparate,D3DXVECTOR3＊ pvecDistance)
//「距離」が「2つの影の合計」より大きい場合（非衝突）はFALSE　等しいか小さい場合（衝突）は
TRUE
// この関数は、OBBCollisionDetection関数のみに呼び出されるサブルーチン
BOOL CompareLength(BBOX＊ pboxA,BBOX＊ pboxB,D3DXVECTOR3＊
pvecSeparate,D3DXVECTOR3＊ pvecDistance)
{
    //”分離軸上での”ボックスAの中心からボックスBの中心までの距離
    FLOAT fDistance=fabsf( D3DXVec3Dot( pvecDistance, pvecSeparate) );
    //分離軸上でボックスAの中心から最も遠いボックスAの頂点までの距離
    FLOAT fShadowA=0;
    //分離軸上でボックスBの中心から最も遠いボックスBの頂点までの距離
    FLOAT fShadowB=0;
    //それぞれのボックスの”影”を算出
    fShadowA = fabsf( D3DXVec3Dot( &pboxA->vecAxisX, pvecSeparate ) * pboxA-
>fLengthX)+
        fabsf( D3DXVec3Dot( &pboxA->vecAxisY, pvecSeparate ) * pboxA-
>fLengthY)+
        fabsf( D3DXVec3Dot( &pboxA->vecAxisZ, pvecSeparate ) * pboxA-
>fLengthZ);

    fShadowB = fabsf( D3DXVec3Dot( &pboxB->vecAxisX, pvecSeparate ) * pboxB-
>fLengthX)+
        fabsf( D3DXVec3Dot( &pboxB->vecAxisY, pvecSeparate ) * pboxB-
>fLengthY)+
        fabsf( D3DXVec3Dot( &pboxB->vecAxisZ, pvecSeparate ) * pboxB-
>fLengthZ);
    if(fDistance > fShadowA + fShadowB)
    {
          return FALSE;
    }
    return TRUE;
}
```

One Point

OBBの詳細については、拙書『ゲームコーディングvol1』（工学社）で解説しています。

249 OBBツリーにより衝突判定する（アニメーションメッシュ）

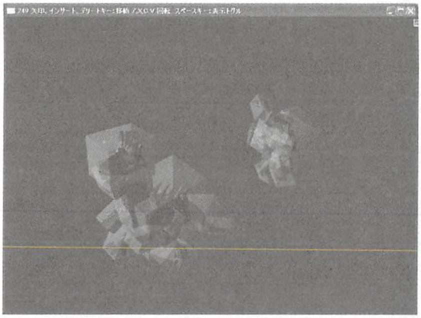
各パーツメッシュ（各フレーム）それぞれに対しOBBを作成する　上級

サンプルプロジェクト　249 OBB Tree

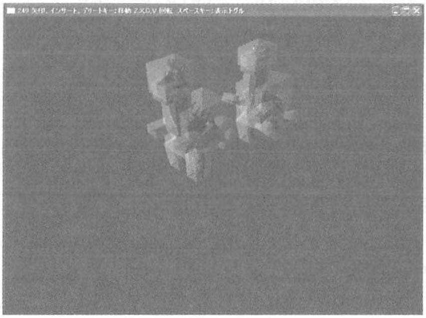
図から明らかなように、各パーツメッシュそれぞれが　バウンディングボックスを持っています。各ボックスはOBBなので、そのメッシュの姿勢に合わせて自由に回転（もちろん移動も）することができます。

各パーツメッシュごとに判定することになるので、むしろ通常のメッシュよりも細かな衝突を検出することができます。

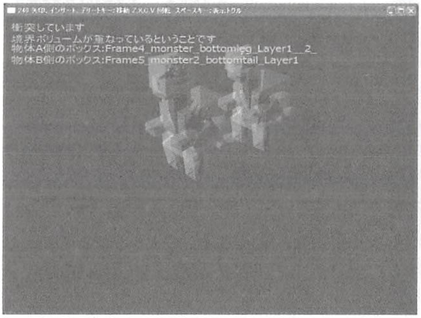
▼実行結果


249 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

▼実行結果


249 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

▼実行結果


249 矢印、インサート、デリートキー：移動 Z,X,C,V 回転 スペースキー：表示トグル

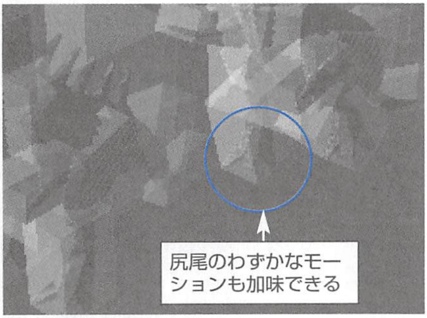
```
衝突しています
境界ボリュームが重なっているということです
物体A側のボックス:Frame4_monster_bottomleg_Layer1__2_
物体B側のボックス:Frame5_monster2_bottomtail_Layer1
```

たとえば、本サンプルの恐竜メッシュは尻尾がゆらゆら揺れています。恐竜メッシュ全体の位置を動かさない場合でも、尻尾は動いているので、当たったり、当たらなかったりという見た目とかなり一致した精度の高い判定となります。

ソースが長いので、ポイントだけ述べます。

OBBの判定部分そのものは直前のサンプルと変わりありません。アニメーションメッシュなので、フレーム階層を走査してゆき、それぞれのノード（フレーム）でやっていることは通常のメッシュと同じでということです(Tips248と同じ)。

▼実行結果



One Point

BBOX構造体をMY_FRAMEに包含すると、もっとシンプルに書けるかもしれません。是非読者自身で実装してみてください。

250 自由落下

重力加速度をかけて自然な落下を表現する

中級

サンプルプロジェクト　250 FreeFall

さて、ここでの自由落下は地球上での落下ですが、空気抵抗や空気粘性を一切考慮しない、言ってみれば真空中での自由落下なので、計算式は最も単純なものになります。初等物理の教科書の最初ページにでも出てきそうな次の式を基にします。

$$y=vt+\frac{1}{2}gt2$$

ただし、yは垂直距離、vは初速度、tは時間、gは重力加速度

…と言いたいところですが、この式すらまじめに使う必要はありません。

このような計算式を使用するとなるとX成分、Z成分の距離も算出しなくてはなりません。そんなことをしなくとも、速度ベクトルを変化させれるだけで簡単にコーディングできます。D3DXVECTOR3型である3次元ベクトルは“速さ”と“方向”を表す“速度”を表すのに便利です。速度はベクトルですから当たり前と言えば当たり前ですね。

本サンプルでも、

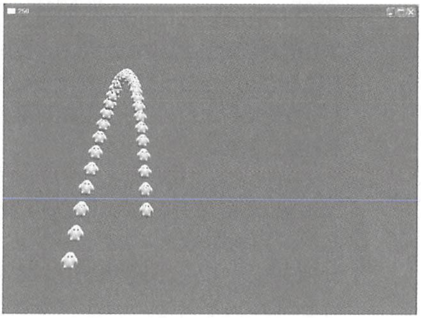
D3DXVECTOR3　g_v(0.08,1,0);//初速度ベクトル

としてg_vというグローバルな速度ベクトルを用意し使用しています。そして、全ての処理はこの速度ベクトルを更新することによりち

ゃんと加速度のついた自由落下を表現します。

初速度ベクトルのx,あるいはz成分が非ゼロであれば、メッシュの軌跡は放物線になります。実際サンプルはそうなります。

▼実行結果



```
D3DXVECTOR3 g_vPos(0,0,0);
D3DXVECTOR3 g_v(0.08,1,0);//初速度ベクトル
```

```
//速度を位置に反映 p=p0+vt; 現在位置=初期位置+速度X時間
g_vPos+=g_v;
//重力をかける（上昇時は重力で速度のy成分が減っていく）
D3DXVECTOR3 vGravity(0,-0.0098 ,0);
g_v+=vGravity;
```

g_vは速度ベクトルです。加速度を毎フレーム加算するので、つねにその時の全体速度を表します。毎フレームごとに位置に速度ベクトルを加算することによりメッシュを移動させます。

重力は鉛直下向きにも働くのでvGravityの成分中、x,z成分はゼロにします。y成分は現実の重力加速度9.806に倣っています。これだけです。

次の部分は、確認しやすいようにリセットしているだけで本質ではありません。

```
//とりあえず位置のy成分がゼロ以下になったら終了
if(g_vPos.y<0)
{
     g_vPos=D3DXVECTOR3(0,0,0);
     g_v=D3DXVECTOR3(0.08,1,0);
}
```

251 跳ね返らせる

地面との反作用を考慮してバウンドさせる

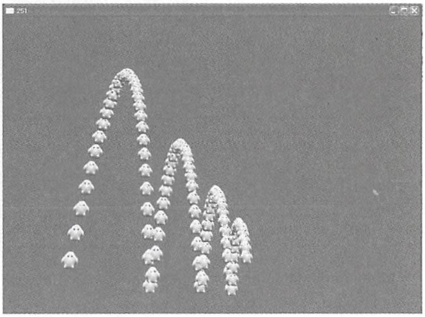
中級

サンプルプロジェクト　251　Bound

直前のTipsに、地面との反作用を加えています。これによりメッシュは、地面に当たると跳ね返されて、また上昇します。そして、重力によりまた落下し、また地面に衝突する…つまりバウンドを繰り返します。

地面と衝突したときに、速度から衝突エネルギーを引かないと、延々と同じ高さでバウンドしてしまうので、差し引きます（現実でもそうです）。

▼実行結果



```
D3DXVECTOR3 g_vPos(0,0,0);
D3DXVECTOR3 g_v(0.08,1,0);//初速度ベクトル
```

```
//速度を位置に反映 p=p0+vt; 現在位置＝初期位置＋速度X時間
g_vPos+=g_v;
//重力をかける（上昇時は重力で速度のy成分が減っていく）
D3DXVECTOR3 vGravity(0,-0.0098 ,0);
g_v+=vGravity;
//とりあえず位置のy成分がゼロ以下になったら地面と衝突したとして処理
if(g_vPos.y<0)
{
    g_vPos.y=0;
    g_v.y*=-1;跳ね返り
    //衝突によるエネルギー損失（空気抵抗等が無いため、損失させないと延々とバウンドしてしまう）
    g_v*=0.8;
}
```

第 3 章

252～303

バッファ、サーフェイス、Xファイルの極意

3-1　メッシュ関連のバッファ（252～272）
3-2　Xファイル（273～281）
3-3　リソース管理、各種サーフェイス（282～303）

252 メッシュ内の頂点バッファにアクセスする

●IDirect3DvertexBuffer9::Lockメソッドにより頂点バッファをロックする 初級

サンプルコード片　252　頂点バッファのロック
参考サンプルプロジェクト　156　テクスチャの無いメッシュにテクスチャを貼る

頂点バッファとは、メッシュオブジェクト内部にあるバッファの一つで、頂点データを格納しているメモリ領域です。

頂点バッファにアクセスすれば、そのメッシュの頂点データを参照したり、時には頂点データを書き換えたりできるわけですが、頂点バッファは特殊なメモリ空間にあるため、そのままではアクセスできません。アクセスするには、バッファをロックしなくてはなりません。コードは次のようになります。

```
/＊
pDeviceはDirect3Dデバイスオブジェクトポインター
pMeshはメッシュオブジェクトのポインター
＊/

//頂点バッファインターフェイスポインターを宣言
LPDIRECT3DVERTEXBUFFER9 pVB=NULL;
//当該メッシュの頂点バッファインターフェイスを取得
pMesh->GetVertexBuffer(&pVB);
//ロックする。成功すればpVerticesは頂点データの先頭アドレスが入って戻ってくる
BYTE＊ pVertices=NULL;
if(SUCCEEDED(pVB->Lock(0,0,(VOID＊＊)&pVertices,0)))
{
     //ここで頂点データの参照なり、書き換えなりを行う

     //アクセスする必要が無くなったらすぐにアンロックする。
     pVB->Unlock();
}
else
{
     MessageBox(0,"頂点バッファーのロック失敗","",MB_OK);
     return E_FAIL;
}
```

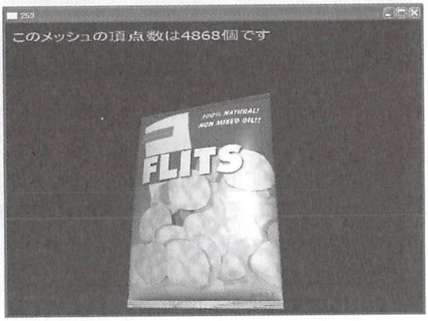
253　メッシュ内の頂点数を取得する

●ID3DXMesh:: GetNumVerticesメソッドにより頂点数を取得する　初級

サンプルプロジェクト　253　メッシュの頂点数

頂点数の取得は簡単で、ID3DXMesh:: GetNumVerticesメソッド1発で取得できます。コードは例えば次のようになります。

▼実行結果


253
100% NATURAL!
NON MIXED OIL!!
FLITS

```
このメッシュの頂点数は4868個です
```

```
DWORD  NumVertices=pMesh->GetNumVertices();
```

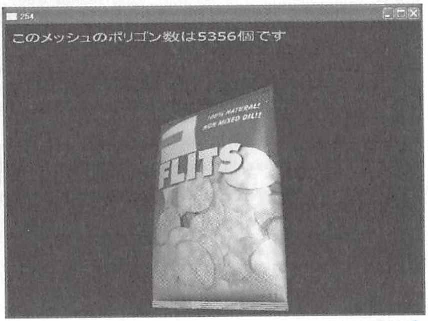
254　メッシュ内のポリゴン数を取得する

●ID3DXMesh:: GetNumFacesメソッドによりポリゴン数を取得する　初級

サンプルプロジェクト　254　メッシュのポリゴン数

ポリゴン数の取得は簡単で、ID3DXMesh:: GetNumFacesメソッド1発で取得できます。コードは例えば次のようになります。

▼実行結果


254
100% NATURAL!
NON MIXED OIL!!
FLITS

```
このメッシュのポリゴン数は5356個です
```

```
DWORD  NumPolys=pMesh->GetNumFaces();
```

255 頂点バッファから頂点情報を取得する

頂点情報のうち座標情報を参照する

中級

サンプルプロジェクト　255　メッシュの頂点情報

まず、ストライドを得ます。ストライドとは頂点ひとつ分のサイズです。本サンプルのメッシュにおける頂点ストライドは32（バイト）で、その内訳は次のとおりです。

座標 x,y,z FLOAT値（4バイト）×3＝12バイト

頂点法線ベクトル x,y,z　FLOAT値（4バイト）×3＝12バイト

テクスチャ座標　FLOAT値（4バイト）×2＝8バイト

合計32バイト。

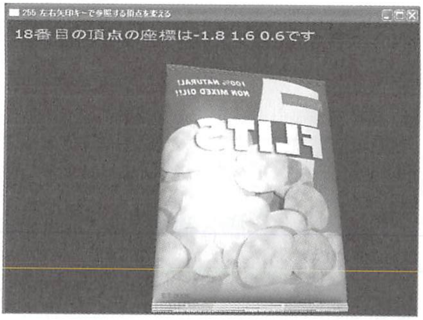
k番目の頂点データのアドレスは、

頂点バッファの先頭アドレズ ＋ (k × ストライド)

となります。

コード上では、pVertices+=g_dwIndex＊dwStride;としてます。pVerticesは頂点データの先頭アドレスですので、それにインデックス×ストライド分を足せば、インデックス番目の頂点データのアドレスとなります。

▼実行結果



18番目の頂点の座標は-1.8 1.6 0.6です

```
LPDIRECT3DVERTEXBUFFER9 pVB=NULL;
pMesh->GetVertexBuffer(&pVB);
DWORD dwStride=pMesh->GetNumBytesPerVertex();
BYTE  ＊pVertices=NULL;
D3DXVECTOR3＊ pvVertex=NULL;
if(SUCCEEDED(pVB->Lock(0,0,(VOID＊＊)&pVertices,0)))
{
      pVertices+=g_dwIndex＊dwStride;
      pvVertex=(D3DXVECTOR3＊)pVertices;
      pVB->Unlock();
 }
```

```
TCHAR sz[100];
sprintf(sz,"%d番目の頂点の座標は%3.1f %3.1f %3.1fです",g_dwIndex,pvVertex-
>x,pvVertex->y,pvVertex->z);
RenderString(sz,10,10);
```

One Point

g_dwIndexはキー操作により増減するグローバル変数ですので、キー操作で参照する頂点座標を変化させることができます。

ロックしたときに頂点データの先頭アドレスを受け取っているのは、あくまでもBYTE＊型のpVerticesです。座標ベクトルとして参照する場合、ベクトルは3成分あり、しかもFLOAT値なのでキャストしなくてはなりません。

pvVertex=(D3DXVECTOR3＊)pVertices;

256 メッシュの頂点に接線情報を加える

接線を含む頂点フォーマットとなるようなメッシュを複製する

上級

サンプルプロジェクト　256　接線を加える

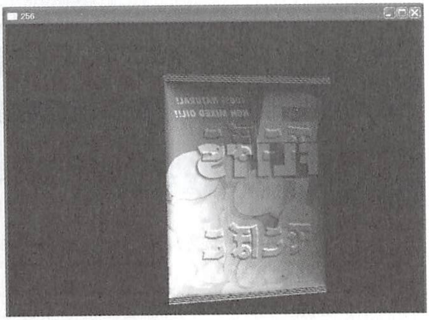
接線（タンジェント）を追加したい局面としては、バンプマッピングするときがあります。

バンプマッピングは法線、接線、従法線の3つのベクトルからポリゴンごとの座標系を必要とします。

しかし、通常のメッシュは接線を持ちませんから、プログラム側で接線を追加しなくてはなりません。

接線を追加すると言っても既存のメッシュに接線情報を挿入するのではなく、実際的に言えば接線を持つようなメッシュを作り直します。

▼実行結果



```
//接線情報を追加
LPD3DXMESH pTmpMesh;
if(FAILED(pMesh->CloneMesh(pMesh->GetOptions(),decl,pDevice, &pTmpMesh )))
{
    MessageBox(NULL, "CloneMesh", NULL, MB_OK);
}
if(FAILED(D3DXComputeNormals( pTmpMesh, NULL )))
{
    MessageBox(NULL, "D3DXComputeNormals", NULL, MB_OK);
}
if(FAILED(D3DXComputeTangent( pTmpMesh,0,      0,
    D3DX_DEFAULT,
    TRUE,  NULL )))
{
    MessageBox(NULL, "D3DXComputeTangent", NULL, MB_OK);
}
SAFE_RELEASE(pMesh);
pMesh = pTmpMesh;
```

CloneMesh関数でメッシュを複製（クローン）しているのが分かると思います、クローンしたメッシュの頂点ストライドは接線の分だけ大きくなります。つまり、接線を置く場所が確保されているとも言えます。そのあとD3DXComputeTangent関数で接線を計算します。

最後に、もはや古いメッシュ（接線の無いメッシュ）は不用なのでリリースし、クローンしたメッシュのポインタを代入すれば完了です。

インデックスバッファ

257 インデックスバッファとは？

インデックスバッファの意味を理解する

初級

「インデックスバッファはポリゴンバッファ」です。

頂点バッファは頂点を格納しているので頂点バッファ、インデックスバッファはインデックスを格納しているのでインデックスバッファ…、実際に格納しているデータで呼ぶと確かにそうなのですが、これでは混乱すると思われます。

そこで、格納しているデータの種類ではなく、そのバッファの機能で呼ぶと、頂点バッファは機能としても頂点バッファですが、インデックスバッファは、「ポリゴンバッファ」と言えます。

頂点バッファ（バーテックスバッファ）に対し、面バッファ（ポリゴンバッファ）と覚えたほうが良いと筆者は思います。

インデックスバッファは任意のポリゴンを形成する3つの頂点（ポリゴンは三角ポリゴンなので頂点は3つです）の番号を保存しているバッファであり、インデックスバッファによりk番目のポリゴンを形成する3つの頂点を簡単に取り出すことができます。

258 インデックスバッファにアクセスする

●IDirect3DvertexBuffer9::Lockメソッドにより頂点バッファをロックする 初級

サンプルコード片 268 LockIndex.cpp
参考サンプルプロジェクト 259 ポリゴンの3頂点

頂点バッファと同様にインデックスバッファにアクセスするには、バッファをロックしなくてはなりません。コードは次のようになります。

```
/＊
pDeviceはDirect3Dデバイスオブジェクトポインター
pMeshはメッシュオブジェクトのポインター
＊/

//インデックスの先頭アドレスを保持するためのポインターを用意
WORD＊ pwPoly=NULL;

//ロック
pMesh->LockIndexBuffer(D3DLOCK_READONLY,(VOID＊＊)&pwPoly);

(なんらかのインデックス参照処理)

  //アクセス不要になったらすぐアンロック
pMesh->UnlockIndexBuffer();
```

259 インデックスによりポリゴンの3頂点を取得する

●任意のポリゴン上の3つの頂点を取り出す 中級

サンプルプロジェクト 259 ポリゴンの3頂点

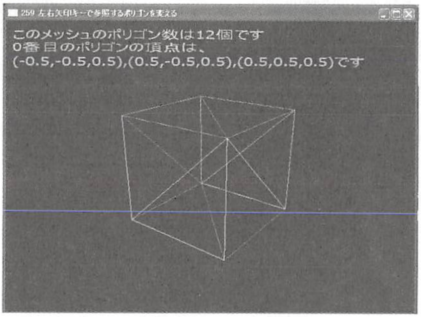
Direct3Dにおける、いわゆるポリゴンは三角ポリゴンのみです。したがってポリゴンの頂点もすべて3つだけであり、頂点数が4つや5つのポリゴンはありません。

このサンプルではインデックスバッファを参照することにより任意のインデックス（番

号）の三角ポリゴンに対する3つの頂点を頂点バッファから取得します。

Direct3Dに、そのような関数が無いのでFindVerticesOnPolyという関数を作成しました。

▼実行結果



このメッシュのポリゴン数は12個です
0番目のポリゴンの頂点は、
(-0.5,-0.5,0.5),(0.5,-0.5,0.5),(0.5,0.5,0.5)です

```
HRESULT FindVerticesOnPoly(LPD3DXMESH pMesh,DWORD dwPolyIndex,D3DXVECTOR3＊
pvecVertices )
{
	DWORD i,k;
	DWORD dwStride=pMesh->GetNumBytesPerVertex();
	DWORD dwVertexAmt=pMesh->GetNumVertices();
	DWORD dwPolyAmt=pMesh->GetNumFaces();
	WORD＊ pwPoly=NULL;
	pMesh->LockIndexBuffer(D3DLOCK_READONLY,(VOID＊＊)&pwPoly);

	BYTE ＊pbVertices=NULL;
	FLOAT＊ pfVetices=NULL;
	LPDIRECT3DVERTEXBUFFER9 VB=NULL;
	pMesh->GetVertexBuffer(&VB);
	if(SUCCEEDED(VB->Lock(0,0,(VOID＊＊)&pbVertices,0)))
	{
		pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex＊ 3]
];
		pvecVertices[0].x=pfVetices[0];
		pvecVertices[0].y=pfVetices[1];
		pvecVertices[0].z=pfVetices[2];

		pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
3+1] ];
		pvecVertices[1].x=pfVetices[0];
		pvecVertices[1].y=pfVetices[1];
		pvecVertices[1].z=pfVetices[2];

		pfVetices=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
3+2] ];
		pvecVertices[2].x=pfVetices[0];
		pvecVertices[2].y=pfVetices[1];
		pvecVertices[2].z=pfVetices[2];
```

```
            pMesh->UnlockIndexBuffer();
            VB->Unlock();
        }
        return S_OK;
    }
```

インデックスバッファは頂点バッファのオフセットでもあります。インデックスバッファのポインタpwPolyの添え字にインデックスを渡せば、頂点バッファ内の当該頂点のアドレスになります。

One Point

Tips233「交差しているメッシュ内のポリゴンを得る」でも、この関数を使用しています。

260 ポリゴンの頂点法線を取得する

頂点データから法線データを取得する

中級

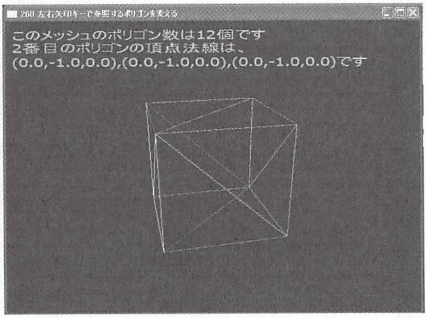
参考サンプルプロジェクト 260 ポリゴンの頂点法線

1つの頂点データ（1つの頂点ストライド内）において法線は座標の後に格納されています。

このサンプルについて留意しなければならないことがあります。このサンプルは特定の頂点タイプにしか機能しません。あくまでも、Box.xメッシュ用のものです。たとえば、頂点データが頂点色を含んでいたり、あるいは頂点ブレンド用の“重み”を格納していたりすると、うまくいきません。

Box.xの頂点データにおいて、法線データは座標データのすぐ後に格納されているのが事前に分かっているので、次のようなある意味“決めうち”コードを書いています。

▼実行結果



```
このメッシュのポリゴン数は12個です
2番目のポリゴンの頂点法線は、
(0.0,-1.0,0.0),(0.0,-1.0,0.0),(0.0,-1.0,0.0)です
```

```
pfNormals=(FLOAT*)&pbVertices[dwStride*pwPoly[ dwPolyIndex* 3]
+sizeof(FLOAT)*3 ];（頂点データの先頭から12バイト目に法線があるとしている）
```

もし、これ以外の頂点フォーマットから法線を取得する場合はコードを改良する必要があります。頂点データを読み込む前に頂点タイプを判別するコードを書けば、対応できるのですが、どうしてもコードが長くなり、また、本質を見失うのであえてこの形にしています。

メッシュ内、任意のポリゴンの頂点法線を取得する関数としてFindNormalsOnPoly関数を作成しました。直前のサンプルにおけるFindVerticesOnPolyとほとんど同じです。異なるのは、12バイト分ポインタをずらしていることだけです。

```
HRESULT FindNormalsOnPoly(LPD3DXMESH pMesh,DWORD dwPolyIndex,D3DXVECTOR3＊
pvNormals )
{
	DWORD i,k;
	DWORD dwStride=pMesh->GetNumBytesPerVertex();
	DWORD dwVertexAmt=pMesh->GetNumVertices();
	DWORD dwPolyAmt=pMesh->GetNumFaces();
	WORD＊ pwPoly=NULL;
	pMesh->LockIndexBuffer(D3DLOCK_READONLY,(VOID＊＊)&pwPoly);

	BYTE ＊pbVertices=NULL;
	FLOAT＊ pfNormals=NULL;
	LPDIRECT3DVERTEXBUFFER9 VB=NULL;
	pMesh->GetVertexBuffer(&VB);
	if(SUCCEEDED(VB->Lock(0,0,(VOID＊＊)&pbVertices,0)))
	{
		pfNormals=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex＊ 3]
+sizeof(FLOAT)＊3 ];
		pvNormals[0].x=pfNormals[0];
		pvNormals[0].y=pfNormals[1];
		pvNormals[0].z=pfNormals[2];

		pfNormals=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
3+1] +sizeof(FLOAT)＊3];
		pvNormals[1].x=pfNormals[0];
		pvNormals[1].y=pfNormals[1];
		pvNormals[1].z=pfNormals[2];

		pfNormals=(FLOAT＊)&pbVertices[dwStride＊pwPoly[ dwPolyIndex ＊
3+2] +sizeof(FLOAT)＊3];
		pvNormals[2].x=pfNormals[0];
		pvNormals[2].y=pfNormals[1];
		pvNormals[2].z=pfNormals[2];

		pMesh->UnlockIndexBuffer();
		VB->Unlock();
```

```
    }
    return S_OK;
}
```

261 ポリゴンの頂点法線から面法線を算出する

ポリゴンの3つの頂点法線から面法線を計算する

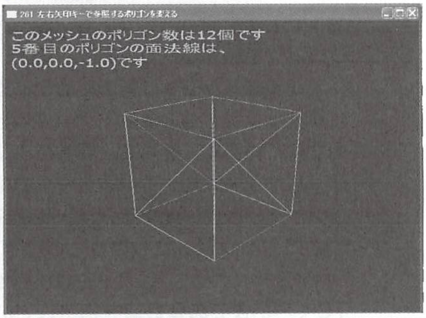
中級

サンプルプロジェクト　261　ポリゴンの面法線

直前のサンプルと頂点法線を取得するところまで全て同じです。異なるのは、文字列表示部分直前で多少の計算をしていることだけです。

3つの頂点法線の単純な加重平均をとって面法線とします。

▼実行結果



```
このメッシュのポリゴン数は12個です
5番目のポリゴンの面法線は、
(0.0,0.0,-1.0)です
```

```
//ポリゴンの参照処理部分
 {
     D3DXVECTOR3 vVN[3];//頂点法線用
     D3DXVECTOR3 vPN;//面法線用
     FindNormalsOnPoly(pMesh,g_dwIndex,vVN);
     //頂点法線の加重平均により面法線を求める
     vPN=(vVN[0]+vVN[1]+vVN[2]) / 3;

     sprintf(sz,"%d番目のポリゴンの面法線は、¥n(%3.1f,%3.1f,%3.1f)です",
                             g_dwIndex,
                             vPN.x,vPN.y,vPN.z);

     RenderString(sz,10,30);
 }
```

属性バッファと属性テーブル

262 属性バッファとは？

●メッシュの属性バッファを理解する

初級

Direct3Dのメッシュオブジェクトは、頂点バッファ、インデックスバッファと、属性バッファの合わせて3つのバッファを持ちます。

頂点バッファは頂点の生データを格納するバッファ、そしてインデックスバッファは構造的には頂点バッファ内の頂点データのオフセットを格納するバッファで、機能的にはポリゴンを格納するバッファでした。

では属性バッファは何のためにあるのか？　属性バッファは、主にマテリアル分けに利用されています。そして、属性バッファの実態（バッファに入っているデータの型）は何なのか？　属性は単なるDWORD値です。

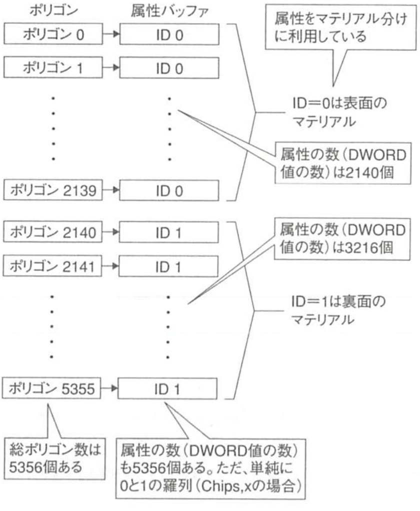
さて、メッシュをレンダリングする際にDrawSubset(i)などとして、0からせいぜい10くらいまでの整数値を指定していたのを思い出してください。その整数値は属性です。

例えばマテリアルが2つあるメッシュの場合、メッシュの全てのポリゴンにつき0あるいは1の属性がつきます。したがって、属性はポリゴンの数と一致します。1000ポリゴンから成るメッシュであれば属性バッファ内の属性(DWORD値)も1000個あります。

本章のサンプル全てで使用するメッシュChips.xは表と裏で合計2つのサーフェイス（マテリアル）を持っています。表側サーフェイスのポリゴン数は2140個で、裏側のポリゴン数は3126個です（この数は実際にTips267のサンプルで表示します）。

次の図にChips.xのサーフェイス（マテリアル）と属性の関係を示しました。

▼属性バッファ

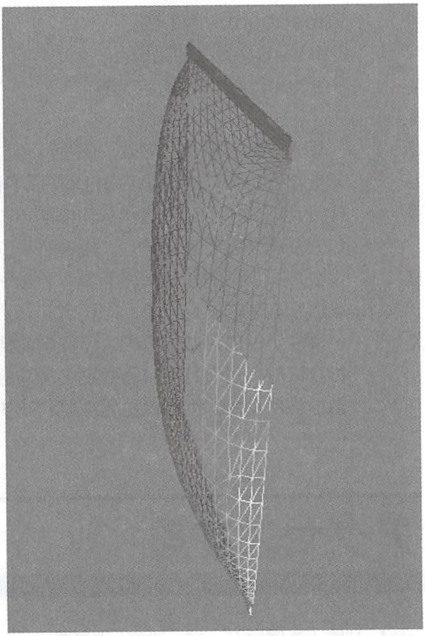

ポリゴン
属性バッファ
ポリゴン 0
ID 0
ポリゴン 1
ID 0
ポリゴン 2139
ID 0
ポリゴン 2140
ID 1
ポリゴン 2141
ID 1
ポリゴン 5355
ID 1
属性をマテリアル分けに利用している
ID＝0は表面のマテリアル
属性の数（DWORD値の数）は2140個
属性の数（DWORD値の数）は3216個
ID＝1は裏面のマテリアル
総ポリゴン数は5356個ある
属性の数（DWORD値の数）も5356個ある。ただ、単純に0と1の羅列（Chips,xの場合）

＊＊＊＊＊＊

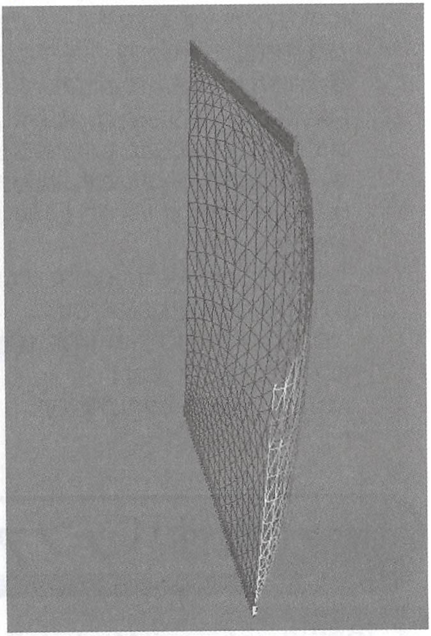
属性バッファの目的について「主にマテリアル分け…」と述べたのは、マテリアル分けに使用しなければならないと決まっているわけではなく、別の用途に使用できるからです。例えばマテリアルが一つだけのメッシュでも、属性を複数持たせることによりポリゴン群に何らかのグループ分けが可能となります。

つまり、属性の目的は「ポリゴン群に何らかのグループ分け（もちろんマテリアル分けも含む）をすることです」。

▼ポリゴン数2140個



▼ポリゴン数3126個



263 属性テーブルとは？

●メッシュの属性テーブルを理解する

初級

属性バッファはDWORD値の単純な羅列であるので、属性を参照・操作するには多少のコードを書かなくてはならないでしょう。

属性0のマテリアルサブセットを抽出するには、例えば次のようなコードを書くでしょう。

```
DWORD＊ pBuffer=（属性バッファの先頭アドレスとする）
DWORD dwNumPoly=0;//マテリアル0のポリゴン数
while( ＊pBuffer ==0)
{
    pBUffer++;
    dwNumPoly++;
}
```

属性バッファは単なる羅列であるので、属性バッファからポリゴングループについて何か情報を得ようとした時、ループにより調べなければなりません。属性に関るポリゴン数、開始ポリゴンインデックス、終了ポリゴンインデックス等をまとめたものがあると便利です。それが属性テーブルです。属性テーブルは、その他に属性グループごとの頂点数情報も持っています。

具体的に属性テーブルはD3DXATTRIBUTERANGE構造体の配列です。

D3DXATTRIBUTERANGE 構造体は次のメンバを持っています。

AttribId	属性テーブルの識別子。
FaceStart	開始面。
FaceCount	面の数。
VertexStart	開始頂点。
VertexCount	頂点の数。

個人的な意見としては、属性バッファの状態をまとめておくためD3DXATTRIBUTERANGEのような構造体を作成することは誰しもが考えるでしょうし、属性バッファの属性グループに関する情報を調べるループコードは初心者でも書けると思います。

属性テーブルは、このような事前調査を代わりにやってくれている程度の意味合いしかありませんが、便利と言えば便利です。

264　属性バッファにアクセスする

●ID3DXMesh::LockAttributeBufferメソッドにより属性バッファをロックする　初級

サンプルコード片　264　属性バッファへのアクセス

属性バッファへアクセスするためには、属性バッファをロックしなくてはなりません。ロックを行うメソッドはメッシュインターフェイスのメソッドです（ID3DXMesh::LockAttributeBuffer() ）。

```
//pMeshはメッシュオブジェクトのポインターとする
DWORD＊ pBuf=NULL;
//属性バッファをロック
pMesh->LockAttributeBuffer(D3DLOCK_READONLY,&pBuf);
//参照処理
(n番目のポリゴンの属性はpBuf[ n ]を参照すれば分かる)
//用事が済んだらすぐにアンロック
pMesh->UnlockAttributeBuffer();
```

参照だけではなく、属性IDを書き換える場合は、LockAttributeBuffer 関数の第1引数にD3DLOCK_READONLYではなくD3DLOCK_DISCARDを指定します。また、ロックの成否に関わらず関数からただちに戻るようにするにはD3DLOCK_DONOTWAITも指定します。

```
LockAttributeBuffer(D3DLOCK_READONLY | D3DLOCK_DONOTWAIT,…
LockAttributeBuffer(D3DLOCK_DISCARD | D3DLOCK_DONOTWAIT,…
```

265 属性テーブルを取得する

ID3DXMesh:: GetAttributeTableメソッドにより属性テーブルを得る　初級

サンプルコード片　265 属性テーブルの取得

属性テーブルを取得するにはID3DXMesh:: GetAttributeTable関数で取得できますが、注意する点は、2段階でGetAttributeTable関数をコールする必要があることです。

最初のコールの目的は、属性の総数だけを得るためです。それを行ってはじめて、本来のコールができます。

```
//pMeshはメッシュオブジェクトのポインターとする
//まずは属性の総数を取得する
DWORD dwNumAttr=0;
pMesh->GetAttributeTable(NULL,&dwNumAttr)；

//その属性分だけメモリ確保
D3DXATTRIBUTERANGE＊ pAttr= new D3DXATTRIBUTERANGE[dwNumAttr];

//メモリ確保した領域に属性テーブルをコピー
pMesh->GetAttributeTable(pAttr,&dwNumAttr)；

//参照処理
(n番目の属性グループはpAttr[ n ]を参照すれば分かる)
```

266 属性バッファを表示する

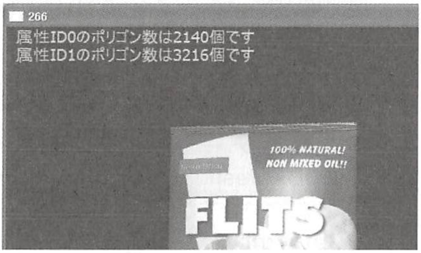
●ID3DXMesh::LockAttributeBufferメソッドでバッファをロックする 初級

サンプルプロジェクト　266　属性バッファの表示

ここでは、属性バッファの内容を実際に表示するということを行います。

バッファの内容を読み取るには、バッファをロックしなくてはなりません。もちろん書き込む際も同様です。

▼実行結果



```
266
属性ID0のポリゴン数は2140個です
属性ID1のポリゴン数は3216個です
```

```
TCHAR sz[200];
DWORD＊ pBuf=NULL;
pMesh->LockAttributeBuffer(D3DLOCK_READONLY,&pBuf);

DWORD dwTotalPoly=pMesh->GetNumFaces();
DWORD dwStartIndex=0;
DWORD dwNumPoly=0;
DWORD dwID=0;

for(DWORD i=0;i<dwTotalPoly;i++)
{

    if(pBuf[i]==dwID)
    {
          dwNumPoly++;
    }
    else
    {
          sprintf(sz,"属性ID%dのポリゴン数は%d個です",dwID,dwNumPoly);
          RenderString(sz,10,dwID＊20);
          dwID++;
          dwNumPoly=1;
    }
}
sprintf(sz,"属性ID%dのポリゴン数は%d個です",dwID,dwNumPoly);
```

```
RenderString(sz,10,dwID＊20);

pMesh->UnlockAttributeBuffer();
```

267 属性テーブルを表示する

●ID3DXMesh:: GetAttributeTableメソッドで属性テーブルを取得する

初級

サンプルプロジェクト 267 属性テーブルの表示

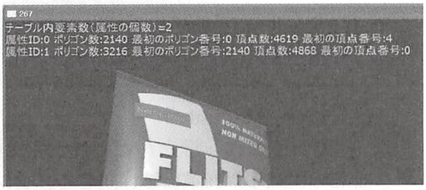
属性バッファそのものを参照するよりも、属性テーブルを参照するほうが多いと思います。

属性テーブルの各要素はD3DXATTRIBUTERANGE型です。そして属性テーブルの要素数は属性の種類だけでとなります。例えば属性ID0が100個、ID1が200個あるような場合でも属性テーブルの要素数は2個となります。

属性テーブルの取得はID3DXMesh::GetAttributeTableメソッドで行います。属性テーブルを取得するにはID3DXMesh::GetAttributeTable関数で取得できますが、注意する点は、2段階でGetAttributeTable関数をコールする必要があることです。

最初のコールの目的は、属性の総数だけを得るためです。それを行ってはじめて、本来のコールができます。

▼実行結果



```
テーブル内要素数(属性の個数)=2
属性ID:0 ポリゴン数:2140 最初のポリゴン番号:0 頂点数:4619 最初の頂点番号:4
属性ID:1 ポリゴン数:3216 最初のポリゴン番号:2140 頂点数:4868 最初の頂点番号:0
```

```
TCHAR sz[200];
DWORD dwNumEntry=0;
pMesh->GetAttributeTable(NULL,&dwNumEntry);
D3DXATTRIBUTERANGE＊ pEntry= new D3DXATTRIBUTERANGE[dwNumEntry];
pMesh->GetAttributeTable(pEntry,&dwNumEntry);
sprintf(sz,"テーブル内要素数（属性の個数）=%d",dwNumEntry);
RenderString(sz,0,0);

for(DWORD i=0;i<dwNumEntry;i++)
{
    sprintf(sz,"属性ID:%d ポリゴン数:%d 最初のポリゴン番号:%d 頂点数:%d 最初の頂点番
号:%d",
pEntry[i].AttribId,pEntry[i].FaceCount,pEntry[i].FaceStart,pEntry[i].VertexC
ount,pEntry[i].VertexStart);
```

```
        RenderString(sz,0,20*i+20);
    }
```

268 属性の活用例(DrawSubsetに依らないサブセットごとのレンダリング)

●DrawIndexedPrimitiveでレンダリングする際の引数に、属性テーブルを利用する

中級

サンプルプロジェクト　268 属性の活用例

ここでは、属性テーブルを手掛かりに、DrawSubsetではなくIDirect3DDevice9::DrawIndexedPrimitiveによりもっと直接的にメッシュのサブセットをレンダリングするということを行います。

Direct3DDevice9::DrawIndexedPrimitiveメソッドは引数で混乱しやすいメソッドです。それ以前に引数の意味が分からない場合も多いかもしれません。

第1引数はポリゴン描画手順のタイプです。これはいいでしょう。

第2引数は、ほとんどの状況では0です。この引数の存在意義は、異なるメッシュ間でインデックスを共有する際の利便のためであり、単一のメッシュでは意味がありません。

第3引数は、最適化に貢献します。ここに全頂点数を指定しても正常にレンダリングされますが、ここに「その属性グループにおける頂点の中で一番若い番号（頂点のインデックス）」を指定することにより、頂点の範囲を限定できます。範囲を指定することにより高速なレンダリングを行わせることができます。

第4引数は頂点数です。頂点数をメッシュの全頂点数にしても問題ないですが、第2引数と同様に、その属性グループにおける頂点数を指定することにより最適化に貢献します。

第5引数がもっとも混乱すると思われます。ここには「インデックスのインデックス」を指定します。インデックスのインデックスとは何を意味するのでしょう？

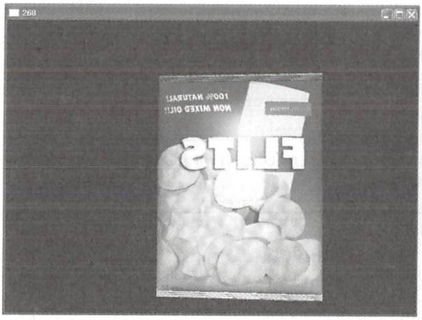
インデックスバッファの各要素は頂点のインデックスなわけですが、インデックスバッファだって配列ですから各要素は「インデックス（並び順）」を持ちます。インデックスバッファ自体の先頭からの要素の番号です。

第6引数はポリゴン数です。

引数の意味が分かれば、属性テーブルがそのまま利用できることが分かるでしょうか？

第3から第6引数なでは、すべて属性テーブルをそのまま使えます（第5引数はポリゴンオフセットに3をかけて頂点オフセットにする必要がありますが）。

▼実行結果



●メッシュ初期化時

```
D3DXATTRIBUTERANGE＊  g_pAttrTable=NULL;
DWORD  g_dwNumAttr=0;
```

```
//メッシュの属性テーブルを取得
pMesh->GetAttributeTable(NULL,&g_dwNumAttr);
g_pAttrTable=  new  D3DXATTRIBUTERANGE[g_dwNumAttr];
```

●毎フレームレンダリング時

```
LPDIRECT3DVERTEXBUFFER9  pVB;
  LPDIRECT3DINDEXBUFFER9  pIB;
  pMesh->GetVertexBuffer(&pVB);
  pMesh->GetIndexBuffer(&pIB);
  pDevice->SetStreamSource(0,  pVB,  0,  pMesh->GetNumBytesPerVertex());

  pDevice->SetFVF(pMesh->GetFVF());
  pDevice->SetIndices(  pIB  );

  for(DWORD  i=0;i<g_dwNumAttr;i++)
  {
      pDevice->SetMaterial(  &pMeshMaterials[i]  );
      pDevice->SetTexture(  0,  pMeshTextures[i]  );

      pDevice->DrawIndexedPrimitive(
             D3DPT_TRIANGLELIST,  //描画タイプ
             0,                //オフセット
             g_pAttrTable[i].VertexStart,       //その属性グループの頂点で最小のイン
デックス
             g_pAttrTable[i].VertexCount,//頂点数（その属性グループの頂点数）
             g_pAttrTable[i].FaceStart*3,  //インデックスのインデックス（つまりインデ
ックスバッファ内のインデックス）
             g_pAttrTable[i].FaceCount);//ポリゴン数
  }

  SAFE_RELEASE(pVB);
  SAFE_RELEASE(pIB);
```

One Point

このサンプルは、DrawIndexedPrimitiveの活用法でもあります。

269 隣接性バッファとは？

隣接性バッファを理解する

中級

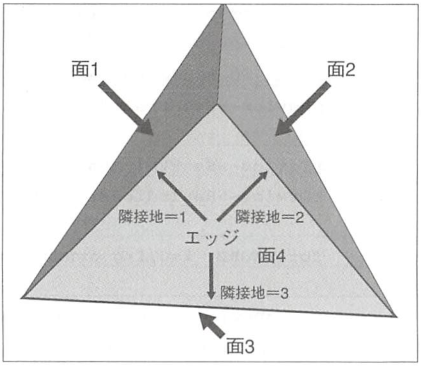
隣接性情報とは、あるポリゴンに隣り合っているポリゴンのインデックスのことです。隣接性バッファとはこの隣接性情報の羅列（バッファ）です。

ポリゴンの辺はエッジとも呼ばれます。Direct3Dは全て三角ポリゴンなので、辺は当然3本です。したがって、一つのポリゴンに対し隣接性情報は3つあります。隣接性情報は32ビット整数（DWORD値）なので、最大で約4億ポリゴンまで対応できることになります。

右の図を見てください。

正面に見える面4に注目すると、面4の隣にある3面は面1、面2、面3となり、面4に関わる隣接性情報は（1,2,3）となります。

▼隣接性情報



270 隣接性バッファの作成

ID3DXMesh::GenerateAdjacencyメソッドにより隣接性バッファを作成する

中級

サンプルコード片　270　隣接性バッファの作成

隣接性情報は1つのポリゴンに対しDWORD値3つとなります。そして、隣接性バッファの要素数はポリゴン数と一致するので、バッファのサイズはポリゴン数×3×4バイト（DWORDのサイズ）となります。

例えば、4ポリゴンから成るメッシュの隣接性バッファのサイズは、3×4×4=48バイトとなります。

さて、隣接性バッファは他の3つのバッファ（頂点、インデックス、属性）とは異なり、メッシュオブジェクトの一部ではありません。隣接性バッファを作成するにはID3DXMesh::GenerateAdjacencyメソッド一発でできますが、バッファそのものはアプリケーション側で用意しなくてはなりません。したがって、GenerateAdjacencyメ

ソッドはバッファの作成というよりは、こちらで用意したバッファに隣接性情報をコピーするメソッドと言ったほうがいいでしょうか。

```
//pMeshはメッシュオブジェクトのポインターとする
DWORD＊  g_pAdjacency=NULL;
//バッファのメモリ確保
g_pAdjacency=new  DWORD[3＊pMesh->GetNumFaces()];
//確保したメモリに隣接性情報をコピー
if(FAILED(pMesh->GenerateAdjacency(0,g_pAdjacency)))
{
    MessageBox(0,"隣接性バッファ作成失敗","",MB_OK);
}
```

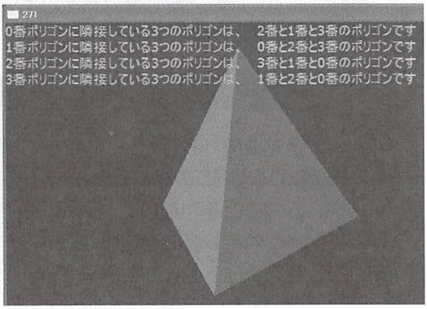
271 隣接性バッファの表示

各ポリゴンに隣り合っている3つのポリゴン番号を表示してみる 中級

サンプルプロジェクト 271 隣接性バッファの表示

コードは単純です。ポリゴン総数だけループし、その中で各ポリゴンに対する隣接ポリゴンのインデックスを表示するという簡単なものです。解説の必要はないでしょう。

▼実行結果



```
271
0番ポリゴンに隣接している3つのポリゴンは、  2番と1番と3番のポリゴンです
1番ポリゴンに隣接している3つのポリゴンは、  0番と2番と3番のポリゴンです
2番ポリゴンに隣接している3つのポリゴンは、  3番と1番と0番のポリゴンです
3番ポリゴンに隣接している3つのポリゴンは、  1番と2番と0番のポリゴンです
```

●**バッファポインタ宣言**

```
DWORD＊ g_pAdjacency=NULL;
```

バッファ作成部分

```
//隣接情報作成
{
    g_pAdjacency=new DWORD[3＊pMesh->GetNumFaces()];
    if(FAILED(pMesh->GenerateAdjacency(0,g_pAdjacency)))
    {
        MessageBox(0,"隣接性バッファ作成失敗","",MB_OK);
        }
    }
```

●表示部分

```
TCHAR sz[200];
for(DWORD i=0,Edge=0;Edge<pMesh->GetNumFaces()*3;Edge+=3,i++)
{
    sprintf(sz,"%d番ポリゴンに隣接している3つのポリゴンは、¥
                    %d番と%d番と%d番のポリゴンです",i,
                    g_pAdjacency[Edge],
                    g_pAdjacency[Edge+1],
                    g_pAdjacency[Edge+2]);

    RenderString(sz,0,i*22);
}
```

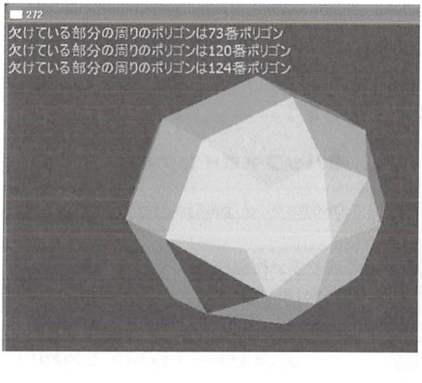
272 隣接性バッファの活用例

●隣接性バッファを、ポリゴンが欠けている部分を調べることに利用する　中級

サンプルプロジェクト　272　隣接性バッファの活用例

エッジに隣り合うポリゴンが無い場合、隣接値はDWORDの最高値0xffffffffになります。このことを利用して、メッシュ内でポリゴンが切れている（ポリゴンが欠けている）部分を見つけることができます。

▼実行結果



```
欠けている部分の周りのポリゴンは73番ポリゴン
欠けている部分の周りのポリゴンは120番ポリゴン
欠けている部分の周りのポリゴンは124番ポリゴン
```

●バッファポインタ宣言

```
DWORD*   g_pAdjacency=NULL;
```

●初期化時　隣接性バッファ作成

```
//隣接情報作成
{
        g_pAdjacency=new  DWORD[3*pMesh->GetNumFaces()];
        if(FAILED(pMesh->GenerateAdjacency(0,g_pAdjacency)))
        {
                MessageBox(0,"隣接性バッファ作成失敗","",MB_OK);
        }
}
```

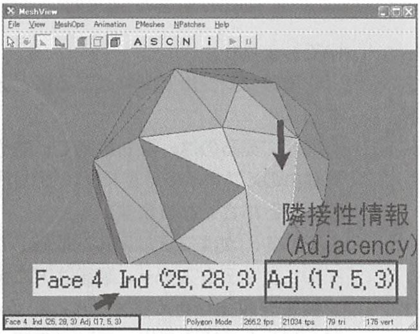
●3面と隣接していないポリゴンの判定

```
for(DWORD  i=0,k=0;i<pMesh->GetNumFaces()*3;i++)
{
        if(g_pAdjacency[i]==0xffffffff)
        {
                sprintf(sz,"欠けている部分の周りのポリゴンは%d番ポリゴン",i/3);
                RenderString(sz,0,k*22);
                k++;
        }
}
```

SDK付属のメッシュビューアーであるMeshView.exeでは隣接値も表示されます。

左下のFace xx Ind (x,x,x) Adj(x,x,x)と表示されている領域があり、Adjが隣接性情報です。ちなみに、Faceはポリゴン番号、Indは頂点インデックス（そのポリゴンを構成する3頂点）です。

▼実行結果

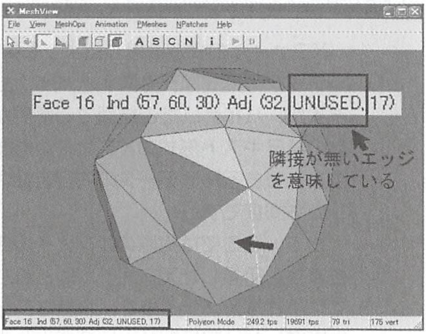

MeshView
File
View
MeshOps
Animation
PMeshes
NPatches
Help
A
S
C
N
i

隣接性情報
(Adjacency)

Face 4 Ind (25, 28, 3) Adj (17, 5, 3)

```
Face 4 Ind (25, 28, 3) Adj (17, 5, 3)    Polygon Mode    266.2 fps    21034 tps    79 tri    175 vert
```

▼実行結果


MeshView
File
View
MeshOps
Animation
PMeshes
NPatches
Help
A
S
C
N
i

Face 16 Ind (57, 60, 30) Adj (32, UNUSED, 17)

隣接が無いエッジ
を意味している

```
Face 16 Ind (57, 60, 30) Adj (32, UNUSED, 17)    Polygon Mode    249.2 fps    19691 tps    79 tri    175 vert
```

One Point

このサンプルは、ポリゴンの欠損を発見するツールに利用できそうです。

273 Xファイルとは？

●Xファイルの意味を理解する

中級

Xファイル（エックス・ファイル）は、メッシュを収録したファイルです。

メッシュにはアニメーションメッシュ、スキンメッシュも含まれます。

メッシュデータはテキスト形式でもバイナリ形式でもどちらでも収録でき、どちらの形式でもプログラム上で読み込むことができます。プログラム上でXファイルを読み込む場合、D3DX系の関数がいくつかあるのでそれを使用するのが一般的です。それらの読み込み関数を使う限り、プログラム上でテキスト、バイナリの違いを意識する必要がありません。関数側で全て処理してくれるので、ソース上での読み込みコードは全く同じものになります。

なお、テクスチャを貼ってあるメッシュの場合、Xファイル内にテクスチャデータ自体を収録することはできないので、別ファイルとして用意します。通常はXファイルと同じ場祖（同じディレクトリ）に置きますが、Xファイル内で画像ファイルを“絶対パス”で記述してしまっている場合は、そのディレクトリに置く必要があります。

274 Xファイルの構造を理解する

●Xファイルの中身を見てみる

中級

Xファイルの構造は思ったよりも単純です。

データの形式はテキストとバイナリがありますが、バイナリだと紙面での解説が不可能なのでテキスト形式のXファイルを覗いてみます。もちろん、データの内容自体はバイナリでも同様です。

さて、ここでは実際のXファイルを見ていきます。用意したXファイルはどれも“極めて”小さなXファイルで、余計な部分（出力に影響しない冗長部分）は全て取り去ったものです。

これほど短いXファイルが、メッシュビューアーで見ることができるれっきとしたXファイルであることに驚く読者もいるのではないでしょうか。

物事はシンプルであればあるほど分かりやすい！

最初に見るのは、最も単純で、おそらく世界最小のXファイルです。（バイナリ圧縮すればもっと小さくなります）

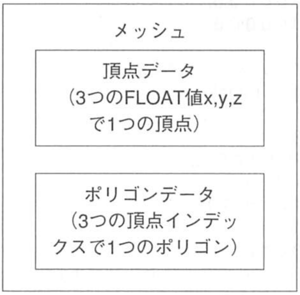
そして、次はマテリアルを収録したXファイル、次はテクスチャ、最後はアニメーション情報を収録したXファイルというように、だんだんと複雑さを増していきます。

Xファイルを実際にメモ帳などで開いて中を見てください。図（の右側）と一致していることが分かります。たったこれだけでXファイルとして成り立つんです。

本書全般に通じる“習うより馴れろ”の精神で中のデータをいろいろ変更して、遊んでみてください。たとえば頂点座標を少しずらしてみるとか、テクスチャファイルを変えてみるとか、アニメーションの位置、サイズを変えてみるとか…etcきっとXファイルの仕組みを把握することができるでしょう。

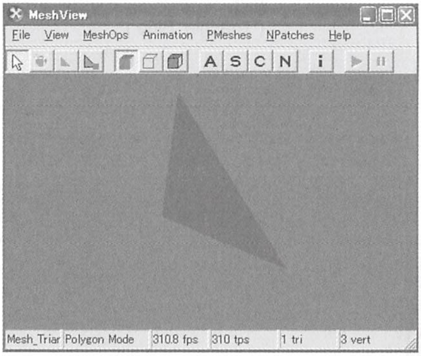
▼マテリアルもテクスチャもない“三角形だけ”のメッシュ

世界最小Xファイル.x


メッシュ
頂点データ
（3つのFLOAT値x,y,z
で1つの頂点）
ポリゴンデータ
（3つの頂点インデッ
クスで1つのポリゴン）

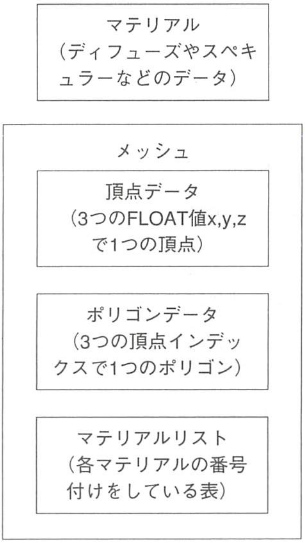
```
x o f 0303txt 0032
////////////////メッシュ////////////////
Mesh Mesh_Triangle
{
        ////////////////頂点データ部////////////////
        3;
        -1.0;-1.0;0.0;,
        -1.0;1.0;0.0;,
        1.0;-1.0;0.0;;
        ////////////////ポリゴンデータ部////////////////
        1;
        3;0,1,2;;
}
```

▼実行結果


MeshView
File
View
MeshOps
Animation
PMeshes
NPatches
Help
A
S
C
N
i
Mesh_Triar
Polygon Mode
310.8 fps
310 tps
1 tri
3 vert

▼マテリアルもテクスチャもない“三角形だけ”のメッシュ

マテリアル付き.x



マテリアル
（ディフューズやスペキュラーなどのデータ）

メッシュ

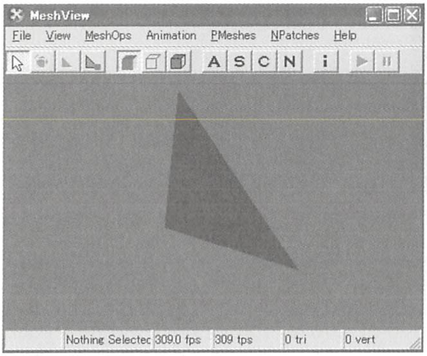
頂点データ
（3つのFLOAT値x,y,zで1つの頂点）

ポリゴンデータ
（3つの頂点インデックスで1つのポリゴン）

マテリアルリスト
（各マテリアルの番号付けをしている表）

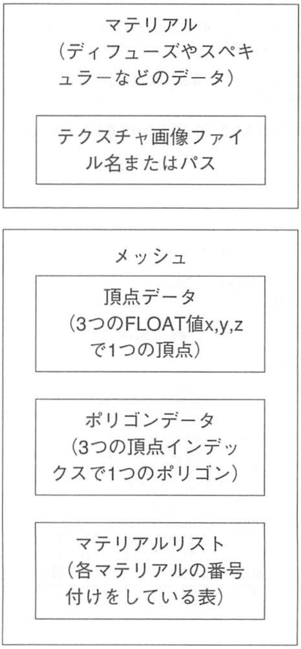
```
x o f 0303txt 0032
///////////////マテリアル///////////////
Material Triangle_Blue
{
          0.0;0.0;1.0;1.0;;
          51.2;
          0.0;0.0;0.0;;
          0.0;0.0;0.0;;
}
///////////////メッシュ///////////////
Mesh Mesh_Triangle
{
          3;
          -1.0;-1.0;0.0;,
          -1.0;1.0;0.0;,
          1.0;-1.0;0.0;;
          1;
          3;0,1,2;;
          ///////////////マテリアルリスト///////////////
          MeshMaterialList
          {
                    1;
                    1;
                    0;
                    { Triangle_Blue }
           }
}
```

▼実行結果


MeshView
File
View
MeshOps
Animation
PMeshes
NPatches
Help
A
S
C
N
i
Nothing Selectec
309.0 fps
309 tps
0 tri
0 vert

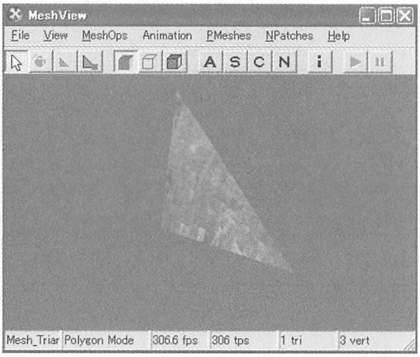
▼コンクリートのテクスチャを貼った三角形

テクスチャ付き.x


マテリアル
（ディフューズやスペキュラーなどのデータ）
テクスチャ画像ファイル名またはパス
メッシュ
頂点データ
（3つのFLOAT値x,y,zで1つの頂点）
ポリゴンデータ
（3つの頂点インデックスで1つのポリゴン）
マテリアルリスト
（各マテリアルの番号付けをしている表）

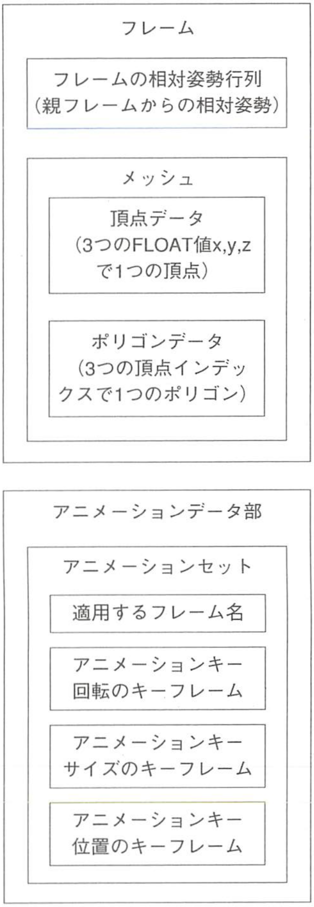
```
x o f 0303txt 0032
///////////////マテリアル///////////////
Material Triangle_Tex
{
        1.0;1.0;1.0;1.0;;
        51.2;
        0.0;0.0;0.0;;
        0.0;0.0;0.0;;
        TextureFilename
        {
                "コンクリート.bmp"
        }
}
///////////////メッシュ///////////////
Mesh Mesh_Triangle
{
        3;
        -1.0;-1.0;0.0;,
        -1.0;1.0;0.0;,
        1.0;-1.0;0.0;;
        1;
        3;0,1,2;;
        ///////////////テクスチャ座標///////////////
        MeshTextureCoords
        {
                3;
                0.0;1.0;,
                0.0;0.0;,
                1.0;1.0;;
        }
        ///////////////マテリアルリスト///////////////
        MeshMaterialList
        {
                1;
                1;
                0;
                { Triangle_Tex }
        }
}
```

▼実行結果


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N
Mesh_Triar
Polygon Mode
306.6 fps
306 tps
1 tri
3 vert

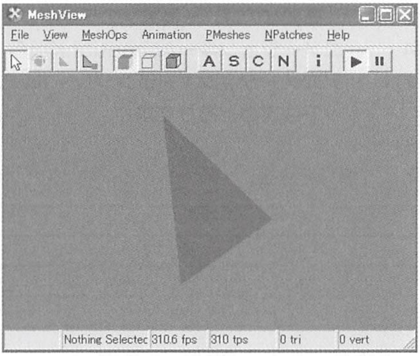
▼横に360度回転する三角形（フレーム数は10）

アニメーション付き.x

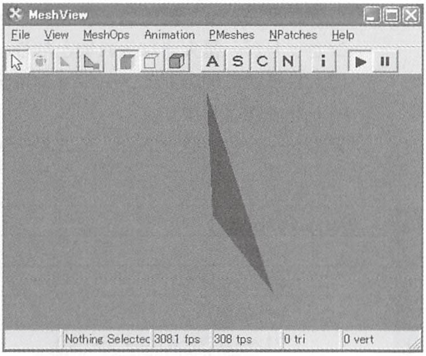

フレーム
フレームの相対姿勢行列
（親フレームからの相対姿勢）
メッシュ
頂点データ
（3つのFLOAT値x,y,z
で1つの頂点）
ポリゴンデータ
（3つの頂点インデッ
クスで1つのポリゴン）
アニメーションデータ部
アニメーションセット
適用するフレーム名
アニメーションキー
回転のキーフレーム
アニメーションキー
サイズのキーフレーム
アニメーションキー
位置のキーフレーム

```
x o f 0303txt 0032
//////フレーム（アニメーションメッシュは単位ではなくフレーム単位）//////
Frame Frame1_ Triangle_ Tex_ Layer1
{
            //////フレームの相対位置（この場合は変化なし）//////
            FrameTransform Matrix
            {
            1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0;;
            }

//////アニメーションデータ部分//////

//////アニメーションセット//////
//////（プログラムから見ると"トラック"）//////
AnimationSet AnimationSet_ Triangle_ Anim
{
            //////このアニメーションは回転のみ//////
            Animation Animation0
            {
                        { Frame1_ Triangle_ Tex_ Layer1 }

                        AnimationKey
                        {
                                    0;//////0は回転のキーフレーム//////
                                    11;
                                    0;4;1.0,0.0,0.0,0.0;;;
                                    1000;4;0.951057,0.0,-0.309017,0.0::,
                                    2000;4;0.809017,0.0,-0.587785,0.0;;,
                                    3000;4;0.587785,0.0,-0.809017,0.0;;,
                                    4000;4;0.309017,0.0,-0.951056,0.0;;,
                                    5000;4;0.0,0.0,-1.0,0.0;;,
                                    6000;4;-0.309017,0.0,-0.951057,0.0;;,
                                    7000;4;-0.587785,0.0,-0.809017,0.0;;,
                                    8000;4;-0.809017,0.0,-0.587785,0.0;;,
                                    9000;4;-0.951056,0.0,-0.309017,0.0;;,
                                    10000;4;-1.0,0.0,-0.0,0.0;;;
                        }
                        AnimationKey
                        {
                                    1;///////1はキーサイズのキーフレーム//////
                                    11;
                                    0;3;1.0,1.0,1.0;;,
                                    1000;3;1.0,1.0,1.0;;,
                                    2000;3;1.0,1.0,1.0;;,
                                    3000;3;1.0,1.0,1.0;;,
                                    4000;3;1.0,1.0,1.0;;,
                                    5000;3;1.0,1.0,1.0;;,
                                    6000;3;1.0,1.0,1.0;;,
                                    7000;3;1.0,1.0,1.0;;,
                                    8000;3;1.0,1.0,1.0;;,
                                    9000;3;1.0,1.0,1.0;;,
                                    10000;3;1.0,1.0,1.0;;,
                        }
                        AnimationKey
                        {
                                    2;//////2は位置のキーフレーム//////
                                    11;
                                    0;3;0.0,0.0,0.0;;,
                                    1000;3;0.0,0.0,0.0;;,
                                    2000;3;0.0,0.0,0.0;;,
                                    3000;3;0.0,0.0,0.0;;,
                                    4000;3;0.0,0.0,0.0;;,
                                    5000;3;0.0,0.0,0.0;;,
                                    6000;3;0.0,0.0,0.0;;,
                                    7000;3;0.0,0.0,0.0;;,
                                    8000;3;0.0,0.0,0.0;;,
                                    9000;3;0.0,0.0,0.0;;,
                                    10000;3;0.0,0.0,0.0;;,
                        }
            }
}
```

▼実行結果


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N i
Nothing Selectec 310.6 fps 310 tps 0 tri 0 vert

▼実行結果


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N i
Nothing Selectec 308.1 fps 308 tps 0 tri 0 vert

●もっと詳しく！

Xファイルの先頭にxof 0303txt 0032と記載されていますが、これは“このファイルはXファイルです”というシグネチャであるので、これは必ず書かなくてはなりません。

各要素（メッシュ、マテリアル、アニメーションセット…等）は“テンプレート”と呼ばれる、いわば“型”が定義されていて、その型の書式に沿って書かれます。

メッシュのテンプレートは次のとおりです。

```
Mesh (メッシュの名前)
{
      (頂点数) ;
      (頂点データ) ;
      (ポリゴン数) ;
      (ポリゴンを構成する頂点3つのインデックス) ;
}
```

これは最低限メッシュであるための型です。メッシュテンプレートは、頂点とポリゴンのほかにその他いろいろなテンプレートを内包することができます。本TipsのXファイルでもマテリアルあるいはテクスチャを貼ったXファイルはマテリアルリストのテンプレートをメッシュテンプレート内に内包しています。

その場合のメッシュテンプレートは、

```
Mesh (メッシュの名前)
{
      (頂点数) ;
      (頂点データ) ;
      (ポリゴン数) ;
      (ポリゴンを構成する頂点3つのインデックス) ;
      (マテリアルリスト)
}
```

となります。

アニメーション部分は、まずアニメーションセットが一番大きな入れ物となり、その中に入れ子状に他のアニメーション系テンプレートを内包させます。

アニメーションセット ＞ アニメーション ＞ アニメーションキー

重要なのは、一番末端のテンプレート、“アニメーションキーAnimationKey”です。アニメーションキーテンプレートのフォーマットは次のとおりです。

```
AnimationKey
{
	（キーのタイプ）
	（キーの数）
	（フレームごとのキーデータ）
}
```

“キーのタイプ”について、0は回転、1はスケール、2は平行移動を意味します。“フレームごとのキーデータ”は“キーの数”だけ記載されることになります。

ここで用意したXファイルは頂点法線を収録していません。法線が無いとグーローシェーディングやライティング計算ができませんので、出力はフラットなものになります。ライティングに耐えうるメッシュにしたい場合は、法線情報をXファイルに追加するか、プログラム側でロード時に法線を追加するなりすることが必要です。メッシュビューアーは後者の方法で法線を追加してくれていると思われます。

275 Xファイル自体はどうやって作るの？

3DCGソフトが必要なことを理解する

初級

Tips 74とTips 101で解説しているとおり、Xファイルそのものは3DCGソフトで作成するしかありません（直前の正解最小Xファイル.xくらいならメモ帳+手書きで作れそうですが）。

筆者の一番のお勧めソフトは“Lightwave3D”です。価格は安いのに、とにかくパワフルで絶対的な性能でも他の3Dパッケージに負けていません。したがってコストパフォーマンスは他の追従を許しません。

Lightwave3DにはXファイルを出力するプラグインが付いてきます。また、SDKおよびファイルフォーマットが公開されているので、自分自身でファイルコンバーター（ライトウェーブファイルからXファイルに変換するツール）を作ることもできます。実際筆者は自身のコンバーターを作成しています。

276 Xファイルのサイズを小さくする

●あくまでファイルのサイズを小さくする方法を覚える

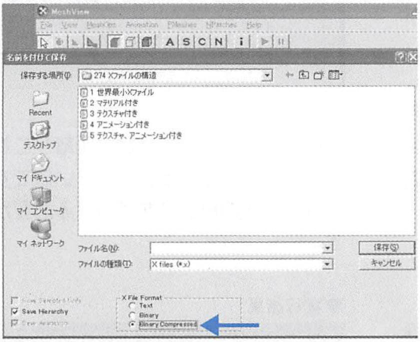
初級

Xファイルのサイズ（バイト数）をコンパクトにするにはメッシュビューアーを使用します。

メッシュビューアーはSDKに無料で付いてツールですので、SDKを持っている人なら誰でも持っていることになります。

まずメッシュビューアーでコンパクトにしたいXファイルを開き、メッシュをクリックしてから（選択してから）、File->Save Mesh As...をクリックすると次のダイアログが出ます。

▼実行結果


MeshView
File View MeshOps Animation PMeshes NPatches Help
A S C N i
名前を付けて保存
Recent
デスクトップ
マイ ドキュメント
マイ コンピュータ
マイ ネットワーク
保存(S)
キャンセル

保存する場所(I): 274 Xファイルの構造

1 世界最小Xファイル
2 マテリアル付き
3 テクスチャ付き
4 アニメーション付き
5 テクスチャ、アニメーション付き

ファイル名(N):
ファイルの種類(T): X files (*.x)

☐ Save Selected Only
☑ Save Hierarchy
☑ Save Animation

X File Format
○ Text
○ Binary
● Binary Compressed

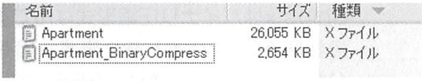
ダイアログボックスの下にX File Formatという領域があり、3つのラジオボタンがあります。そのうち一番下の「Binary Compressed」を選択します。これで「保存(S)」を押せば圧縮されたバイナリーXファイルができ上がります。

実際の例を示しましょう。

Apartment.xはテキスト形式で、なんと26メガバイト！もあります。Xeon2.4、QuadroFXマシンでも開くのに20秒かかります。

これをバイナリ圧縮すると、サイズは10分の1の2.6メガになり、メッシュビューアーでも3秒で開けるようになりました。

▼実行結果



名前	サイズ	種類
Apartment	26,055 KB	X ファイル
Apartment_BinaryCompress	2,654 KB	X ファイル

277 メッシュの頂点数を削減してXファイルを小さくする

●メッシュのジオメトリ自体をコンパクトにして、その結果ファイルも小さくする　初級

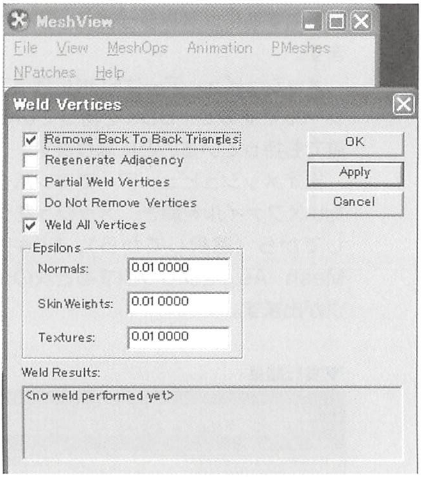
Tips276では、あくまでもジオメトリはそのままで、ファイルを圧縮することによりサイズを小さくしましたが、ここではメッシュのジオメトリ自体をコンパクトにします。具体的には頂点数を減らします。Tips276はファイルのサイズを変えただけであって、プログラムで読み込んでしまえば、同じものとなってしまい、実行速度上の恩恵はありません。

それに対し、このようにジオメトリ自体を削減すると、読み込んだあとのメモリも節約できますし、実行速度も速くなる、言ってみれば本当の意味でのコンパクト化です。

それにより、見かけが多少異なる可能性がありますが、それは見かけと削減量のバランスを考え個々の状況により判断してください。Chips.xはバイナリ圧縮形式なんですが、その状態で96Kバイトあります。図を見れば分かるように、頂点数が4868、ポリゴン数が5356あります。

この頂点数を減らしてみましょう。

▼実行結果


MeshView
File View MeshOps Animation PMeshes NPatches Help
Weld Vertices
OK
Apply
Cancel

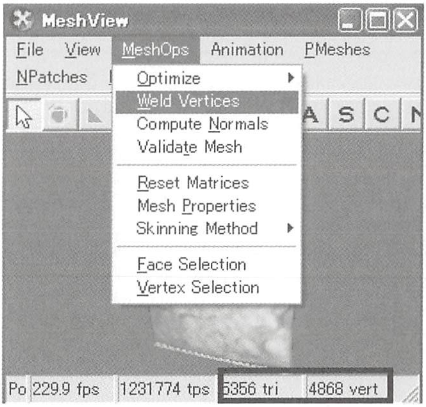
☑ Remove Back To Back Triangles
☐ Regenerate Adjacency
☐ Partial Weld Vertices
☐ Do Not Remove Vertices
☑ Weld All Vertices

Epsilons
Normals: 0.010000
SkinWeights: 0.010000
Textures: 0.010000

Weld Results:
<no weld performed yet>

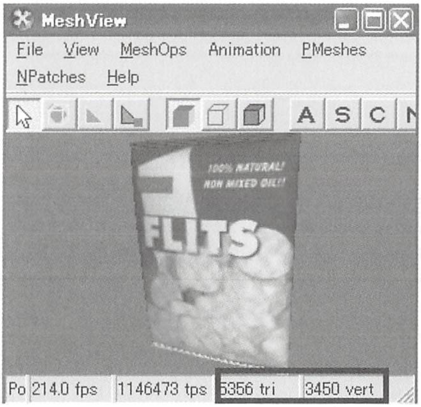
Epsilonsは、「頂点数の距離がこれ以下の場合に結合する」というしきい値です。適当にバランスの良い値を見つける必要があります。

▼実行結果


MeshView
File View MeshOps Animation PMeshes
NPatches
Optimize
Weld Vertices
Compute Normals
Validate Mesh
Reset Matrices
Mesh Properties
Skinning Method
Face Selection
Vertex Selection
A S C

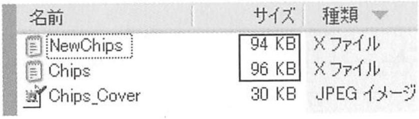
Po 229.9 fps　1231774 tps　5356 tri　4868 vert

▼実行結果


MeshView
File View MeshOps Animation PMeshes
NPatches Help
A S C
100% NATURAL!
NON MIXED OIL!!
FLITS

Po 214.0 fps　1146473 tps　5356 tri　3450 vert

▼実行結果



名前	サイズ	種類
NewChips	94 KB	Xファイル
Chips	96 KB	Xファイル
Chips_Cover	30 KB	JPEG イメージ

278 テクスチャが貼られない場合の原因は？

●Xファイルとテクスチャ用画像ファイルの関係を把握する

初級

テクスチャ付きのメッシュのはずなのに、Xファイルを開いてもテクスチャが貼られないという状況になった場合、次の原因が考えられます。

【原因その1】

テクスチャになる画像ファイルが無い。

【対処】

これは、明らかにテクスチャが貼られるわけがありません。テクスチャ画像ファイルをXファイルと同じディレクトリに置いてください。なお、Xファイル内でテクスチャを絶対パスで指定している場合は、そのディレクトリに置かなければなりません。

【原因その2】

画像ファイルをXファイルと同じディレクトリに置いていても、X　ファイル内でのパスが他のディレクトリに固定されてしまっている。

例えば、次のようになってしまっている。

```
TextureFilename
{
    "C:¥¥Program Files¥¥ゲーム制作¥画像作成中¥¥コンクリート.bmp";
}
```

【対処】

Xファイルのパスを相対パスにする。最も簡単なのはファイル名のみにすることです。

```
TextureFilename
{
    "コンクリート.bmp";
}
```

Xファイルや画像ファイルが大量にあって、ゲームの実行ファイル等とは別の画像フォルダに詰め込んでしまいたいときは、例えば次のように相対パスにします。

```
TextureFilename
{
    "..¥¥画像フォルダ¥¥コンクリート.bmp";
}
```

279 同じメッシュでもポリゴン数が違うのはなぜ？

●Direct3Dのポリゴンと3DCGのポリゴンの違いを把握する

初級

3DCGソフトで三角ポリゴンしかサポートしていないものはありません（4大パッケージしか知りませんが）。

4角ポリゴン、そしてそれ以上の多角形ポリゴンをサポートしています。それに対し、Direct3Dは三角ポリゴンのみです。そのため、3Dソフト上では4頂点（以上）で一つのポリゴンを表現している場合、Direct3Dよりもポリゴン数が少なくなります。

表現を変えると、Xファイルに変換される際に、それらの多角形ポリゴンは3角形に分割されますから、ポリゴン数が増えます。

したがって、3DCGソフト上で三角形に分割すればXファイルにした時のポリゴン数を予想できます。

280 Xファイルにするとポリゴンが欠けたりするのはなぜ？

●Xファイルに変換した後での不具合に対処する

初級

3DCGソフト上ではポリゴンが正常なのにXファイルに変換するとポリゴンが裏返ったり、欠けたりすることがたまにあります。これは、一重に3DCGソフトが4頂点以上の多角形ポリゴンをサポートしていることに起因します（直前のTipsを参照してください）。Xファイル変換の際、4角形以上のポリゴンを3角形に分割する過程で起こります。

対処としては、簡単です。3DCG上の段階で3角形に分割し、その状態にしてからXファイル化すればいいのです、こうすれば、不本意な分割が起きるはずがありません。

281 Xファイルに複数のアニメーション情報を収録する

●アニメーショントラックを手動で複数収録する 中級

Xファイル変換ツールが、複数のアニメーションセット（プログラム上はアニメーショントラック）を出力できない場合、どうすればいいでしょう？

簡単です。メモ帳などで開いて、アニメーションセット部分をコピーアンドペーストすればいいだけです。したがってこれができるのはテキスト形式のXファイルだけですが、サイズが気になるなら、すべて終わった段階でバイナリにすればいいだけの話です。

アニメーションセットを複数持つことにより、モーションのブレンドが可能になります。

例えば、最初は「歩く」モーションしか無かったXファイルの末尾に、他のXファイルから「ジャンプする」アニメーションセットのみをペーストします。このXファイルを読み込んで作成されたアニメーションメッシュは、それぞれのモーションを再生できることはもちろん、「歩きながらジャンプ」するという新たなモーションを再生することができます。これがモーションブレンドであり、複数のアニメーションセットを収録したXファイルのアドバンテージです。

サーフェイス

282 プールとは？

●頂点、ポリゴン、テクスチャ、サーフェイスを格納するメモリ 中級

“プール”とは、頂点、ポリゴン（頂点のインデックス）、テクスチャ、サーフェイスを格納する「メモリ」のことです。メモリにはビデオメモリとシステムメモリがありますから、プールはその両方を意味します。

なお、ビデオメモリにはさらにローカルビデオメモリとAGPメモリがあります。

ローカルビデオメモリは文字通りビデオカード上のオンボードメモリです。AGPメモリは物理的実態としてはシステムメモリと同じPC内のメモリです。AGPメモリという第3のメモリが物理的に独立してあるわけではありません。あくまでその物理的実態はPC内のメモリです。つまりPC内のメモリはシステムメモリになったりAGPメモリになったりします。物理的実態は同じで、そのアクセス方法を変えることにより特性を変えるわけです。

AGPメモリはプログラム上（ドライバ上）から見れば、“ビデオメモリ”になり、ビデオカード側から高速なアクセスができる代わりに、逆にシステムからはアクセスできなくなります。文字通り“ビデオメモリ”として振る舞い、ある意味でローカルビデオメモリ

と同じです。

さて、このように物理的メモリは2種類、論理メモリも2種類ありますが、AGPメモリの関係で完全に1対1に対応しているわけではないことを覚えておきましょう。

283 プールの種類とそれぞれの意味は？

●ビデオメモリとシステムメモリの相反する性質を理解する

中級

Direct3Dにおいて、頂点、ポリゴン（頂点のインデックス）、テクスチャ、サーフェイス（この4つをここではリソースと表現します）を格納するプール（メモリ）をどっちのメモリ（ビデオかシステムか）にするかというリソースの格納先（と管理方法）について4つの方法があります。

それらを見ていく前に覚えておくことがあります。

Direct3DデバイスつまりGPU（CPUではない）はビデオメモリにアクセスできますが（当然ですね）、逆にシステムメモリにはアクセスできません。したがってリソースがシステムメモリだけにあるような場合はレンダリングしようとしても失敗します（何も表示されません）。

一方、CPU（プログラム側）はビデオメモリにはアクセスできませんが、システムメモリにあるリソースには“ロック”することによりアクセスできます。

このことを踏まえてプールとリソースの関係を見ていきましょう。

●D3DPOOL_SYSTEMMEM

システムメモリにリソースを格納します。GPUがアクセスできないので、そのリソースのみを使用するようなレンダリングは失敗します。

ではなぜ、このようなメモリクラスがあるのか？

システムメモリは、GPUからアクセスできない代わりにCPU（プログラム側）からアクセスできるとうメリットがあります。

また、デバイスを消失したときでもリソースは破壊されないので、再読み込みする必要がありません。レンダリングに直接は使えませんが、ビデオメモリに“コピー”することはできます。この性質により、用途としては、リソースの“原型保存用”とでも言いますか、いつでも正常なリソースを引き出すことができる安心した格納場所として使えます。

実際、メッシュは読み込み直後はシステムメモリ上に格納されます。メッシュのレンダリングが成功するのは、システムメモリからビデオメモリにデータをコピーしているからです。

●D3DPOOL_DEFAULT

ビデオメモリにリソースを配置します。ビデオメモリはGPUからアクセスできるので、レンダリングは成功します（当たり前ですが）。

ただし、ビデオメモリですからCPU（プログラム）からはアクセスできません。（“ロック”できません）

●D3DPOOL_MANAGED

これは、D3DPOOL_SYSTEMMEMとD3DPOOL__DEFAULTの“良いとこ取り”のようなメモリクラスです。ビデオメモリにリソースを配置するので、当然そのリソースをレンダリングできます。さらに、システムメモリにもそのリソースのコピーをバックアップするので、ロックができたり、デバイスを消失したときでもリソースを作り直す必要がありません。

また、ビデオメモリ上のリソースの開放も自動的に行ってくれるなど、まさに“管理”を代行してくれるようなメモリクラスです。

●D3DPOOL_SCRATCH

システムメモリにリソースを配置します。D3DPOOL_SYSTEMMEMに似ていますが、D3DPOOL_SYSTEMMEMがビデオメモリにデータを転送できるのに対し、このメモリクラスは何もできません。ただし、リソースのロックができるのでアクセスはできます（アクセスもできないとしたら存在価値が全くないですよね）。

ではなぜ、D3DPOOL_SYSTEMMEMとは別にこのメモリクラスがあるのか？

レンダリングするためではなくリソースのデータ解析など、ツール系プログラムのための保管場所というのが用途でしょうか。

284 ロックとは？

●ロックするということの意味を理解する

中級

Tips283で述べたように、システムメモリにリソースを置くとプログラム側からアクセスできます。

ただ、アクセスできるといっても、アプリケーションで普通に用意した配列やメモリ領域へのアクセスほどの自由度はありません。プログラムで用意したメモリ領域には何も考えずにポインターでそのままアクセスできますが、そのようなことまではできません。

システムメモリ上のリソースへアクセスするためには、“ロック”しなくてはなりません。そして、用事が済んだらすぐに“アンロック”しなければいけません。ロック中は本来の操作ができなかったりパフォーマンスが落ちるので、ロックは必要最小限にすることが大切です。なお、リソースがビデオメモリにある場合は、これもTips283で述べたようにプログラム側からはアクセスできないので、ロックすらできません。

ロックは大抵の場合、そのリソースインターフェイスのLockメソッドで行い、リソースインターフェイスにはLockメソッドが用意されています。

例えば、頂点バッファについて見てみると、メッシュの頂点バッファにアクセスする場合は頂点バッファインターフェイスのLockメソッドをコールします。

```
LPD3DXMESH pMesh;
LPDIRECT3DVERTEXBUFFER9 pVB;
. . . . . . . . . . . . . . .
pMesh->GetVertexBuffer( &pVB );
VOID＊ pVertices = NULL;
pVB->Lock( 0, 0, &pVertices, 0 );
(ロック中の処理)
pVB->Unlock();
```

この方法では、事前のそのメッシュの頂点バッファインターフェイスを取得する必要があります。これが面倒な場合を想定してか、メッシュの場合はメッシュインターフェイスから直接頂点バッファ等をロックすることができます。

```
VOID＊  pVertices  =  NULL;
pMesh->LockVertexBuffer(D3DLOCK_READONLY,&pVertices);
(ロック中の処理)
pMesh->UnlockVertexBuffer();
```

インデックスバッファ、属性バッファも同様です。

```
DWORD＊  pIndices  =  NULL;
pMesh->  LockIndexBuffer(D3DLOCK_READONLY,(VOID＊＊)&pIndices);
(ロック中の処理)
pMesh->  UnlockIndexBuffer();
```

各ロックメソッドの第1引数は、ロックのモードを意味します。第2引数にはそれぞれのリソースの先頭アドレスが入って帰ってきます。

頂点の場合は、FLOAT値（座標など）もあればDWORD値（頂点色など）もあるのでVOID型のポインターを用意します。そのため引数に渡すときはVOID＊＊型にキャストする必要はありませんが、使用するときにはそれぞれの型にキャストする必要があります。

一方、インデックスや属性は型が最初からDWORDしかないことが分かっているので、DWORD型のポインターを用意することができます。そのため第2引数に渡すときはVOID＊＊型にキャストしなければコンパイルエラーを起こします。しかし、使用時にキャストする必要はなくなります。
ここでは第1引数を全て“読み込み専用”であるD3DLOCK_READONLYにしています。第1引数の意味については、次のTipsを参照してください。

“ロック”が必要なリソースとメソッドは次のとおりです。

```
IDirect3DCubeTexture9::LockRect
IDirect3DIndexBuffer9::Lock
IDirect3DSurface9::LockRect
IDirect3DTexture9::LockRect
IDirect3DVertexBuffer9::Lock
IDirect3DVolume9::LockBox
IDirect3DVolumeTexture9::LockBox
ID3DXBaseMesh::LockIndexBuffer
ID3DXBaseMesh::LockVertexBuffer
ID3DXMesh::LockAttributeBuffer
ID3DXPatchMesh::LockAttributeBuffer
ID3DXPatchMesh::LockIndexBuffer
ID3DXPatchMesh::LockVertexBuffer
```

285 ロック時のフラグの意味は？

●ロックモードの意味を理解する

中級

“読み込み専用なのか書き込み専用なのか”という意味のフラグは次の2つで、お互いに排他的です（2つを同時に指定できません）。

D3DLOCK_READONLY…読み込み専用。

D3DLOCK_DISCARD…書き込み専用。

● “ロック処理そのもの”の速度を上げる、つまりロックメソッドがすぐに戻ってくるようにするフラグ

D3DLOCK_DONOTWAIT…ロックの成否に関わらず、すぐに処理が戻ります。ただし、ロックができない場合は戻り値D3DERRR_WASSTILLDRAWINGを返すので、戻り値を調べてそれに対処するコードを書く必要が出てきます。

D3DLOCK_NOOVERWRITE…間接的にロックメソッドの速度（ロックがアプリケーションに制御を戻す速度）を上げます。これは、上書きしないことを通知すればロックは多少速くなるという性質があるからです。

●“ロック中“に他のシステム（マウス等）への影響を少なくするフラグ。他のウィンドウサブシステムに迷惑を掛けないようにするフラグ

```
D3DLOCK_NOSYSLOCK
```

●ダーティ領域（ダーティ領域についてはTips286を参照してください）を変更したくないときのフラグ

D3DLOCK_NO_DIRTY_UPDATE…このフラグを指定しないでロックするとロックした領域がダーティとしてマークされてしまいます。サーフェイスやテクスチャに設定されていたダーティ領域を変化させたくないときにこのフラグを指定します。

お互いに排他関係にあるフラグは同時に指定できませんが、例えば書き込みモードで、かつ、ロックからすぐに帰ってくるようにしたい場合は、

```
Lock(D3DLOCK_DISCARD                  |
D3DLOCK_DONOTWAIT , ・・・);
```

などとします。

テクスチャ（サーフェイスのコンテナ）

286 ダーティ領域とは？

●ダーティ領域の意味を理解する

中級

Direct3Dではサーフェイス（およびテクスチャ内のサーフェイス）について、ダーティ領域という概念を用いています。ダーティ領域の“ダーティ”とは何でしょう？

ダーティ:Dirtyは直訳すれば“汚れた”という意味ですが、コンピューターの世界では“更新された”あるいは、“更新されるべき”という意味で使用されています。もともとはファイルシステムに関る分野での用語であり、Direct3D独自の概念ではありません。

ファイルシステムに関るのは、テキストエディタ、複数の人間でファイルとディスクを共有するサーバー環境もそうですし、もちろん皆さんが使っているウィンドウズそのものもそうですから、コンピューターにとってファイルシステムは根幹を成すシステムであり、したがってダーティという概念は実は昔から重要な概念です。

例えば、メモ帳、ワード、あるいはVisual C++のソースエディタもそうですが、開いている文書（またはソースコード）に、新たに何かを書き加えると、ファイル名の横などにアスタリスク（＊マーク）が付きますよね、あれこそが“ダーティ”になったことの印です。その文書が変更され更新しなければならないことになるわけですが、その変更が反映されるのはユーザーがファイルを保存するときですよね？

言い方を変えると、ユーザーが保存するまで、その変更は画面上（メモリ上）だけの話で、ディスク上の文書にはなんら変更は反映されないことになります。このことはダーティの本質でもあります。つまり、ダーティとマークされた瞬間にその変更が即座に反映されるのではなく、反映されるのは別のタイミングということです。

Direct3Dにおけるサーフェイスのダーティ領域もこれと全く同じ挙動をします。Direct3Dのサーフェイスの更新も、変更があったときではなく、パイプラインのある決まったタイミングで一括して変更されます。

つまり、

UpdataSurface関数などでサーフェイスを更新→即座にサーフェイスが更新
ではなく
UpdataSurface関数などでサーフェイスを更新→ダーティ領域が更新される→ダーティ領域をもとにサーフェイスが更新
という2段構えになっています。

言い換えるとソース上での更新は間接的な更新になります。間接的なという意味はつまり、パイプラインにより固定的な流れ作業に入る前に、プログラム側でいろいろな変更箇所を事前にマーキングしておくしかないということです。Direct3Dはパイプライン主導ですから、間接的な更新はサーフェイスに限ったことでありません。

このように個々の変更時点で変化を反映させずに、ある特定のタイミングで一括して変更を反映させる理由は、もちろん全体パフォーマンスを上げるため（低下させないため）です。

ちなみにSQLサーバーの世界では、変更が加えられたページ（変更を反映させなければならないページ）をダーティページと呼びますし、そもそももっと昔からウィンドウズに限らずファイルシステムの分野ではファイル内で変更があった部分をダーティ部分と呼び、そのファイルをダーティファイルと呼んでいます。

One Point

サーフェイスはなにも指定しないとサーフェイス全体がダーティ領域になっているので（サーフェイスの全部分が更新対象）、ダーティ領域のことを意識することがないと思います。

Column

ダーティ領域を“汚れた領域”と覚えてもいい

プログラム側：関数A君が部屋（サーフェイス）を汚して（変更して）しまった、あっ関数B君も…、面倒臭いから片付け（更新）は君がしておいてよ。

パイプライン側：ナンデヤネン！
プログラム側：僕らがいちいち片付けると時間かかるよ～へっへ
パイプライン側：ウッ…、ソレハコマリマス。
プログラム側：でしょ。汚れた部分だけ印を付けて（ダーティとしてマークして）おくから、そんなに大変じゃないよ。まとめてやったほうが速いでしょ。
パイプライン側：ゴシンセツニ　ドウモ
プログラム側：ところでいつ片付けるの？
パイプライン側：ツギノカタヅケハ　ツギノレンダリングノトキ　ニナリマス
プログラム側：そう。君はいつも流れ作業で決まったことしかやらないから、速いよね。その代わり一旦作業に入ったら何も聞こうとしないから先にいろいろ伝えなきゃならないのがおちゃめだけど。ははは
パイプライン側：（このくそガキャー）
プログラム側：ん？　今なんか言った？　しかも平仮名だったような…
パイプラインの苦悩は続く…

287 ダーティ領域を設定する

●AddDirtyRectメソッドにより意図したダーティ領域を設定する　中級

サンプルプロジェクト　287 ダーティ領域

テクスチャにダーティ領域を設定して、その効果を確認してみましょう。ダーティ領域の追加はIDirect3DTexture9::AddDirtyRect関数で行います。

ダーティ領域は通常、サーフェイスの全体ですが、サーフェイスの一部分だけを変更する場合、そしてその変更部分が分かっている場合は、ダーティ領域をその部分として指定してやれば、無駄な更新部分を省くことができます。

```
LPDIRECT3DTEXTURE9  pSourceTexture=NULL;
```

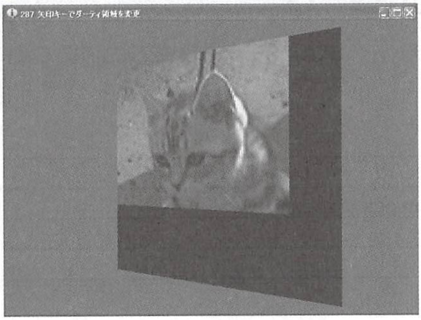
```
RECT Rect;
Rect.top=0;
Rect.left=0;
Rect.right=U;
Rect.bottom=V;
pSourceTexture->AddDirtyRect(&Rect);
```

変数UとVはグローバルに宣言したFLOAT値で、プロシージャーでキー入力により変更できるようにしています。

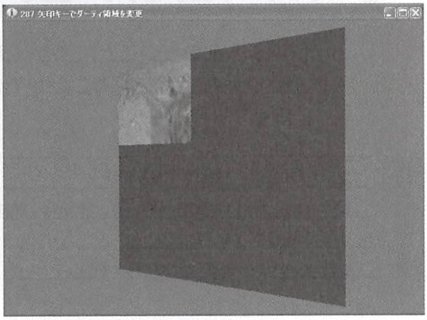
```
FLOAT  U=50,V=50;
```

```
case WM_KEYDOWN:
    switch(wParam)
    {
    case VK_LEFT:
            U-=(U>1);
            break;
    case VK_RIGHT:
            U+=1;
            break;
    case VK_UP:
        V-=(V>1);
        break;
    case VK_DOWN:
        V+=1;
        break;
    }
```

▼実行結果


287 矢印キーでダーティ領域を変更

▼実行結果


287 矢印キーでダーティ領域を変更

なお、Direct3Dには通常のテクスチャの他にはキューブテクスチャとボリュームテクスチャがあります。

それらのテクスチャではメソッド名は次のとおりです。

IDirect3DCubeTexture9::AddDirtyRect
IDirect3DVolumeTexture9::AddDirtyBox

OnePoint

ダーティ領域はビデオカードによってはサポートされないものもあるそうです。その場合はロックとコピーでエミュレートされるようで、その場合パフォーマンスは大幅に低下すると思われます。

288 テクスチャをロックする

LockRectメソッドでテクスチャをロックする

中級

サンプルプロジェクト　288 テクスチャのロック

テクスチャをロックことにより、テクスチャ内のピクセル情報を得ることができます。また、ダーティ領域を追加することにもなります。

```
D3DLOCKED_RECT  Rect;
pTexture->LockRect(0,&Rect,NULL,D3DLOCK_DISCARD);
(なんらかの処理)
pTexture->UnlockRect(0);
```

Rectには、そのテクスチャ内のピクセルへのポインタとピッチが格納されて帰ってきますので、ピクセルを操作するときに使えます。

第2引数にNULLを指定するとテクスチャ全体をロックします。一部分のみのロックでいい場合は、ここにRECT型の矩形を指定します。例えば次のように。

```
D3DLOCKED_RECT  Rect;
RECT  R={0,0,100,100};
pTexture->LockRect(0,&Rect,&R,D3DLOCK_DISCARD);
```

第3引数の意味については、Tips285を参照してください。

なお、ロックするテクスチャは基本的にシステムメモリになければなりません（その理由についてはTips282、283、284を参照してください）。

したがって、テクスチャの作成時D3DXCreateTextureFromFileEx関数の第8引数はD3DPOOL_SYSTEMMEMかD3DPOOL_MANAGEDでなければなりません。特にそのテクスチャをレンダリングに使用する場合はビデオメモリにもなければならないので、そのような場合はD3DPOOL_MANAGEDしか選択肢はありません（ビデオメモリのテクスチャロックについてはTips289を参照）。

```
D3DXCreateTextureFromFileEx(pDevice,"Tex.jpg",0,0,1,0,D3DFMT_R8G8B8,D3DPOOL_MANAGED,D3DX_FILTER_NONE,1,0xff000000,NULL,NULL,&pTexture);
```

289 ビデオメモリにあるテクスチャをロックする

●D3DUSAGE_DYNAMICフラグによりビデオメモリ上のテクスチャでもロックできるようにする　中級

サンプルプロジェクト　289　ビデオメモリテクスチャのロック

ビデオメモリ上のリソースはロックできないのが原則ですが、テクスチャのD3DUSAGE_DYNAMICで作成することにより例外的にロックできるようになります。

```
D3DXCreateTextureFromFileEx(pDevice,"Tex.jpg",0,0,1,
            D3DUSAGE_DYNAMIC,//使用目的をUSAGE_DYNAMICにするとビデオメモリ上のテクスチャでもロックできる

D3DFMT_R8G8B8,D3DPOOL_DEFAULT,D3DX_FILTER_NONE,1,0xff000000,NULL,NULL,&pTexture);
```

290 テクスチャ内のピクセルを変更する

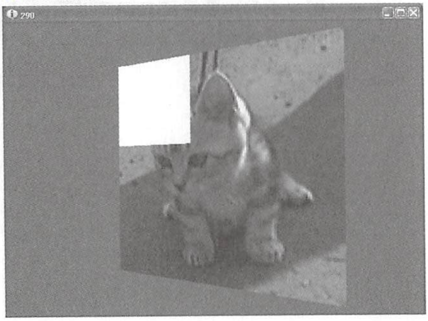
●テクスチャをロックして内部のピクセルを操作する　中級

サンプルプロジェクト　290　テクスチャ内のピクセル

テクスチャをロックする目的のほとんどは、テクスチャ内のピクセルを参照したり変更するためでしょう。

コードは次のようになります。

▼実行結果



```
//アクセスするためにロックする
D3DLOCKED_RECT Rect;
pTexture->LockRect(0,&Rect,NULL,D3DLOCK_DISCARD);
//ピクセルを操作
BYTE* pPixel=(BYTE*)Rect.pBits;

for(DWORD i=0;i<50;i++)
{
    DWORD* pLinePixel=(DWORD*)pPixel;//24ビットであることが分かっているのでピクセルのポインターをDWORDで宣言できる
    for(DWORD k=0;k<50;k++)
    {
            *pLinePixel=0xffffffff;//白にする
            pLinePixel++;
    }
    pPixel+=Rect.Pitch;
}

pTexture->UnlockRect(0);
```

このサンプルではテクスチャのピクセルを変更するので、LockRectメソッドの第3引数はD3DLOCK_DISCARDにする必要があります（ロックフラグについてはTips285を参照）。

ロックが成功すれば、第1引数に渡した矩形にピクセルへのポインタとピッチが入って帰ってきます。あとは、そのピクセルポインターによりピクセルを更新すればいいだけです。

注意しなければならないことは、かならず横幅の計算にはピッチを使うことです。

```
pPixel+=Rect.Pitch;
```

ピッチの重要性についてはTips142を参照してください。

291 テクスチャ内のピクセルを参照する

●テクスチャをロックして内部のピクセルを参照する　中級

参考サンプルプロジェクト　290　テクスチャ内のピクセル

直前のサンプルと同様です。行う処理は直前のTipsよりも単純でしょう。

```
D3DLOCKED_RECT  Rect;
pTexture->LockRect(0,&Rect,NULL,D3DLOCK_READONLY);
//ピクセルのアドレス
BYTE＊  pPixel=(BYTE＊)Rect.pBits;
```

参照の場合は、D3DLOCK_READONLYで十分です。そうしたほうが多少速くロックできるはずです。

ピクセルデータのアドレスは、RectのpBitsメンバに格納されますので、あとはそれを使って自由に参照します。

292 テクスチャからテクスチャへピクセルをコピーする

●UpdateTextureメソッドによりテクスチャ間でデータを転送する　中級

参考サンプルプロジェクト　287　ダーティ領域

Direct3DデバイスオブジェクトのUpdateTextureメソッドでテクスチャ間のデータ転送を行います。

```
pDevice->UpdateTexture(pSourceTexture,pTargetTexture)
```

引数はもちろんテクスチャオブジェクトのポインターです。

```
LPDIRECT3DTEXTURE9  pSourceTexture=NULL;
LPDIRECT3DTEXTURE9  pTargetTexture=NULL;
```

注意することは、ソース側（第1引数）はシステムメモリ、転送先（第2引数）はビデオメモリである必要があります。両方ともビデオメモリにおきたい場合は、Direct3DデバイスオブジェクトのStretchRectメソッドを使用するしかありません。

ただ、StretchRectは制限がけっこう厳しい上、機能も乏しいことから、エフェクト部分には向いていないと思われます。

293 テクスチャを単色で塗りつぶす

●ColorFillメソッドによりテクスチャを塗りつぶす

中級

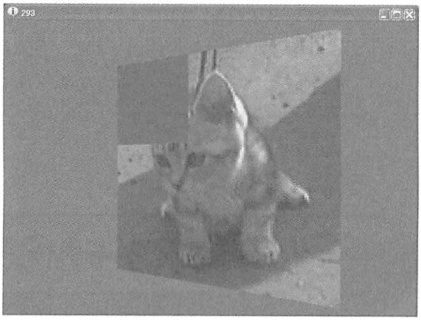
サンプルプロジェクト　293 テクスチャを塗りつぶす

単色での塗りつぶしはDirect3DデバイスオブジェクトのColorFillメソッドで行えます。

テクスチャのピクセルを変更するという意味でTips290と似ていますが、ColorFillは単色にしか変更できません。ただ、逆に言うと、変更する色が単色で良い場合はこの方法のほうがはるかに高速です。

さて、テクスチャと言ってもピクセルを格納しているのはサーフェイスであって、テクスチャはサーフェイスを格納しているコンテナでしかありません。ColorFillはサーフェイスに対して行うものであるので、テクスチャ内のサーフェイスを取得することから始めます。

▼実行結果



```
//テクスチャ内のサーフェイスを得る
LPDIRECT3DSURFACE9  pSurface;
pTexture->GetSurfaceLevel(0,&pSurface);
//そのサーフェイスを塗りつぶす
RECT  Rect;
Rect.top=0;
Rect.left=0;
Rect.right=50;
Rect.bottom=50;
if(FAILED(pDevice->ColorFill(pSurface,&Rect,D3DCOLOR_ARGB(255,0,0,255))))
{
        MessageBox(0,"ColorFill失敗","",MB_OK);
        PostQuitMessage(0);
    }
    SAFE_RELEASE(pSurface);
```

一時的に取得したサーフェイスポインタは、開放しないと参照カウントがどんどん溜まってメモリーリークを起こしてしまいますので、必ずリリースします。

```
    SAFE_RELEASE(pSurface);
```

第2引数は、どの部分を塗りつぶすかを表す矩形です、ここでは50×50ピクセル、開始点は(0,0)に指定しています。ここをNULLにするとサーフェイス全体を塗りつぶします。

COMインターフェイスの参照カウント関連については拙書『ゲームコーディングvol.1』（工学社）で詳細に解説しています。

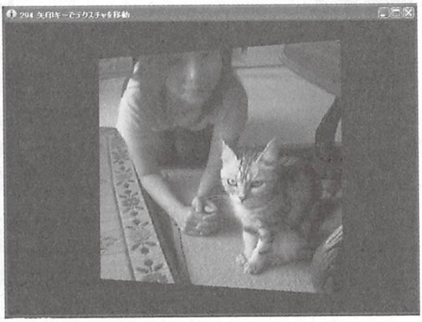
294 テクスチャ内のイメージを動かす

テクスチャをロックすることなく絵を移動する

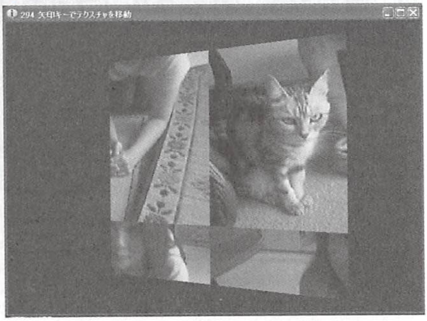
中級

サンプルプロジェクト　294　テクスチャの絵を移動

▼実行結果



▼実行結果



Direct3D9にはテクスチャを移動する仕組みが提供されています。もしそれを使用せずに自前でロック→ピクセル移動なんてやっていたら、おそらくこの100倍くらいの時間がかかると思います。

コードは次のようになります。

```
//テクスチャ移動処理
{
        D3DXMATRIX mat;
        D3DXMatrixIdentity(&mat);
        mat._31=U;
        mat._32=V;
        pDevice->SetTransform(D3DTS_TEXTURE0, &mat);
        pDevice->SetTextureStageState(0, D3DTSS_TEXTURETRANSFORMFLAGS,
D3DTTFF_COUNT2);
}
//テクスチャをセット
pDevice->SetTexture(0,pTexture);

(レンダリング処理)

//テクスチャ操作モードを解除
pDevice->SetTextureStageState(0, D3DTSS_TEXTURETRANSFORMFLAGS,
D3DTTFF_DISABLE);
```

画面バッファ関連（サーフェイス）

295 フロントバッファ、バックバッファとは？

●フロント、バックバッファの意味を理解する

初級

フロントバッファとは、“今ディスプレイに表示されている”内容を保持しているビデオメモリー上のサーフェイスです。言い方を変えれば、“今ディスプレイに表示されている”ピクセルはフロントバッファの内容です。このような意味でフロントバッファ＝ディスプレイ画面　と言えます。

バックバッファとは、フロントバッファに表示する直前のピクセルを置いておく場所です。バックバッファについて、これでは分からないと思うので、もう少し詳しく説明します。

なぜバックバッファが必要なのか？

もしフロントバッファだけしか無い場合、意図した絵（ピクセル）が表示されることはもちろんされるでしょうが、前回（前のフレーム）の絵が残っているのに現在の絵が描画されていくことになり、画面がちらついてしまいます。これをテアリングと言います。しかし、バックバッファがあることによりテアリングを解消することができます。どういうことかと言うと、絵を直接フロントバッファに転送するのではなく、バックバッファに転送します。バックバッファに絵が転送されきった時点でフロントバッファとバックバッファを切り替えます（これをスワップあるいは

フリップと言います）。これを繰り返すことにより、前フレームの絵と現在フレームの絵が同時に画面に表示されるということが無くなり、滑らかな表示が可能になります。このようにバックバッファで1クッションおくことによりテアリングを無くす手法は広く行われていることです。Direct3Dでは、このようなフロント・バックの切り替えをバックバッファリングと呼んでいます。

また、バックバッファをバックバッファリングの側面から見た時、特にスワップ チェーンと呼んでいます。デフォルトではバックバッファは一つなので、その場合スワップチェーン内のバックバッファは一つですが、Direct3Dはバックバッファを2つ以上持たせることができます。

さて、フリップは一般的にフロントとバックのアドレスを切り替えるだけで、中身を入れ替えるわけではありません。それにより高速な切り替えを行います。したがってあるフレームでフロントだったサーフェイスは次のフレームではバックとなり、逆にバックだったサーフェイスが今度はフロントとして振る舞うわけです。絶対的なフロントバッファ、バックバッファが存在すえうわけではないことを理解しましょう。

Direct3Dにおいて留意しなければならないのは、スワップが一般的なフリップと同義になるのは基本的にフルスクリーンのときだけです。ウィンドウモードのときは、アドレス切り替えではなく、フロントとバック間でピクセルの転送が伴い、これがフルスクリーンよりパフォーマンスが落ちる理由でもあります。

296 深度バッファ（Zバッファ）とは？

●Zバッファの意味を理解する

初級

深度バッファあるいはZバッファと呼ばれるものは、ピクセルのZ深度、すなわち各ピクセルについての画面手前から画面奥に向かった距離を保存するバッファのことです。Direct3Dデバイスにはフロントバッファと少なくとも一つのバックバッファを持たせますが、このZバッファを持たせるかどうかは任意であり、不用な場合、たとえば、前後の判定が不要な単純な2Dスプライトを表示するだけの場合、Zバッファを持たせなくても構いません。

ただ、通常は2Dスプライトのみのプログラムであってもピクセルの前後関係を反映しなければならないことがほとんどですし、3Dシーンに至ってはピクセルの前後関係が判定できないとまともなレンダリングが行えませんので、Zバッファはたいてい持たせます。

Zバッファがあるお陰で、ポリゴンの陰面処理が正常に行えます。もしZバッファが無かったら画面に対するポリゴンの前後関係を知る手段がないので、メッシュで本来後ろにあるポリゴン面が手前に表示されてしまうなど、正常なレンダリングができなくなります。

297 ステンシルバッファとは？

●ステンシルバッファの意味を理解する　中級

ステンシル:Stencilとは、「型枠」という意味であり、ステンシルバッファを直訳通り「型枠」バッファと考えても構いません。

型枠バッファなのは分かるけど、なんのために使うの？という声が聞えてきそうです。

ステンシルバッファは、特殊なエフェクトを実装する場合に重宝するバッファであり、逆に言うとそのような特殊な効果のために生まれたバッファとも言えるでしょう。ステンシルバッファがキーファクターとなるエフェクトとして、シャドウボリュームがあります(Tips　218～220参照)。

あるいは、特定の領域のみレンダリングすることにより“鏡”を作成したりする場合などはステンシル（型枠）そのものです。通常のサーフェイスをステンシルバッファとし機能させることは可能と言えば可能ですが、エフェクトの多くはピクセルベースでの操作なので、処理ステップ数は膨大なわけで、事実上使い物になりません。ハードウェアレベル（あるいはドライバレベル）で実装されている必要があり、そのような要請によりステンシルバッファが生まれたわけです。

レンダリングターゲット

298 レンダリングターゲットを変更する

●IDirect3DDevice9::SetRenderTargetメソッドでレンダリングターゲットを変更する。　中級

参考プロジェクト　169,203,215,216,222.225

レンダリングターゲットの変更は、IDirect3DDevice9::SetRenderTargetメソッドで行います。

例えば次のようになります。

```
pDevice->SetRenderTarget(0,pSurf  );
```

第1引数は、レンダリングターゲットが複数ある場合（マルチモニター）におけるレンダリングターゲットのインデックスですが、マルチモニターでもない限り、普通は0です。第2引数は、新たにレンダリングターゲットにするサーフェイスのポインターです。サーフェイスですから、テクスチャ（内のサーフェイス）もレンダリングターゲットになり得

ます。実際、ポストプロセス系のサンプルは全て、テクスチャをレンダリングターゲットにしてエフェクトを実現しています（Tips 215、216、222、225参照）。

Tips169では、キューブテクスチャをレンダリングターゲットにしています。Tips203では、複数のバックバッファを用意し、それぞれをレンダリングターゲットとして切り替えながら分割レンダリングを実現しています。

299 テクスチャ上にレンダリングする

●IDirect3DDevice9::SetRenderTargetメソッドでテクスチャをレンダリングターゲットにする 上級

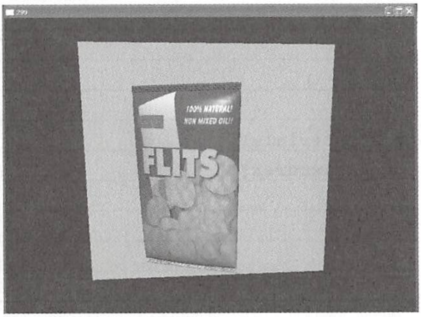
サンプルプロジェクト　299　テクスチャにレンダリング

シーンは通常、モニターに対して、言い換えるとバックバッファーにレンダリングするわけですが、時としてテクスチャ上にレンダリングしたい場合があります。2Dのポストプロセスを行う場合は必ずと言っていいほどテクスチャへのレンダリングを行います。

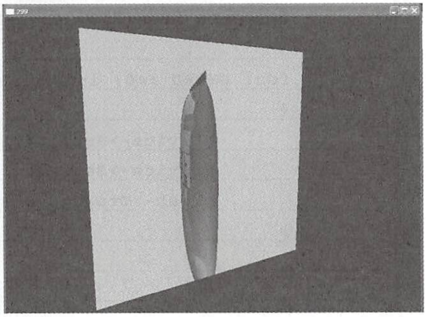
その他にテクスチャにレンダリングしたい状況としては、“テレビ”を実現したい場合などもあります。ブラウン管の部分にテクスチャを対応させておいて、シーンをそのテクスチャにレンダリングすればテレビが動いたとしても、ブラウン管部分に動きのある絵を貼るということが容易に実現できます。

次の図を見てください。板ポリゴンに貼ってあるテクスチャ上にChipsをレンダリングしているわけですが、板ポリゴン自体の回転と、板ポリゴン上の絵（Chips）も回転しているのがわかります。

▼実行結果



▼実行結果



テクスチャにレンダリングするには、レンダリングターゲットをテクスチャに変更します。それに伴って、本来のレンダリングターゲットであるバックバッファを保存しておくことも必要になりますが、仕組みとしてはこれだけです。

実際のコードは次のようなコードになります。

やっていることは、まずレンダリングターゲットを変更するまえに、バックバッファとZバッファを保存しておきます。これを行わないと通常のレンダリングに戻したいときに戻せなくなってしまいます。

次にDirect3DデバイスインターフェイスのSetRenderTargetメソッドによりレンダリングターゲットをテクスチャに指定します。より厳密に言えばテクスチャ内のサーフェイスを取得して、そのサーフェイスをレンダリングターゲットに指定します。

あとは、通常どおりのレンダリングを行えば、テクスチャにレンダリングすることになります。

```
LPDIRECT3DSURFACE9  pBackBuffer  =  NULL;
LPDIRECT3DSURFACE9  pZBuffer  =  NULL;
LPDIRECT3DSURFACE9  pTextureZBuffer  =  NULL;
```

```
// シーンのレンダリング レンダリング先は”テクスチャ”
 {
     //通常時のバックバッファ、Zバッファを保存していく（後でもとのレンダリングターゲットに戻す際に必要）
     pDevice->GetRenderTarget(0,&pBackBuffer);
     pDevice->GetDepthStencilSurface(&pZBuffer);
     //レンダリングターゲットをテクスチャにセット
     LPDIRECT3DSURFACE9 pSurf = NULL;
     pTexture->GetSurfaceLevel(0,&pSurf);
     pDevice->SetRenderTarget(0,pSurf );
     pDevice->SetDepthStencilSurface(pTextureZBuffer);
     SAFE_RELEASE(pSurf);

     pDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER ,
     D3DCOLOR_XRGB(200,200,100), 1.0f, 0 );

     for( DWORD i=0; i<dwNumMaterials; i++ )
     {
            pDevice->SetMaterial( &pMeshMaterials[i] );
            pDevice->SetTexture( 0, pMeshTextures[i] );
            pMesh->DrawSubset( i );
     }

 }
```

なお、本来であればここまでで目的は達成されるのですが、本サンプルではテクスチャに本当にレンダリングされているのか確認する意味で、そのテクスチャ自身をレンダリングして可視化しています。

可視化するので、もちろんレンダリングターゲットは通常のバックバッファに対し行います。そのためには、テクスチャに設定して

いたレンダリングターゲットをバックバッファに戻す必要があります。

```
 //テクスチャのレンダリング　レンダリング先は”バックバッファ”
 {
     //レンダリングターゲットを元に戻す（バックバッファに戻す）
     pDevice->SetRenderTarget(0,pBackBuffer );
     pDevice->SetDepthStencilSurface(pZBuffer);
     SAFE_RELEASE(pBackBuffer);
     SAFE_RELEASE(pZBuffer);

     pDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER ,
     D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );

     TEX_VERTEX vecPoint[4+1]=
     {
             { -2,2,0,0,0 }, //頂点1
             {  2,2,0,1,0 }, //頂点2
             { -2,-2,0,0,1 },//頂点3
             { 2,-2,0,1,1 }  //頂点4
     };
     pDevice->SetTexture(0,pTexture);
     D3DXMatrixRotationY( &matWorld, timeGetTime()/3000.0f );

     pDevice->SetTransform( D3DTS_WORLD, &matWorld );
     pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
     pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX1);
     pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint,
sizeof(TEX_VERTEX));
     pDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
 }
```

300 空のオフスクリーンサーフェイスを作成する

●IDirect3DDevice9::CreateOffscreenPlainSurfaceメソッドによりオフスクリーンサーフェイスを作成する

初級

参考プロジェクト　301,302,303

例えば、サイズが800×600、ピクセルフォーマットがアルファつき32ビットのサーフェイスをシステムメモリーに作成するコードは次のようになります。

```
LPDIRECT3DSURFACE9 pSurface=NULL;
pDevice->CreateOffscreenPlainSurface(
                    800,
                    600,
                    D3DFMT_A8R8G8B8,
                    D3DPOOL_SYSTEMMEM,
                    &pSurface,
                    NULL);
```

また、デスクトップと同じサイズ、かつデスクトップのピクセルフォーマットで作成する場合は、次のようになります。

```
LPDIRECT3DSURFACE9 pCopyFrontBuffer=NULL;
D3DDISPLAYMODE DesktopMode;
pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,&DesktopMode);
pDevice->CreateOffscreenPlainSurface(
     DesktopMode.Width,
     DesktopMode.Height,
     DesktopMode.Format,
     D3DPOOL_SYSTEMMEM,
     &pCopyFrontBuffer,
     NULL);
```

301 フルスクリーン時に表示されているピクセルデータを取得する

IDirect3DDevice9::GetFrontBufferDataメソッドによりフロントバッファの内容を得る

上級

サンプルプロジェクト　301　フルスクリーンのピクセル取得

画面に表示されているそのままのピクセルを取得したい場合があります。その場合は、画面すなわちフロントバッファの内容が取得できればいいことになります。Direct3Dではフロントバッファにアクセスすることは禁止されていますが、フロントバッファの内容を適当な場所（サーフェイス）にコピーすることはできます。

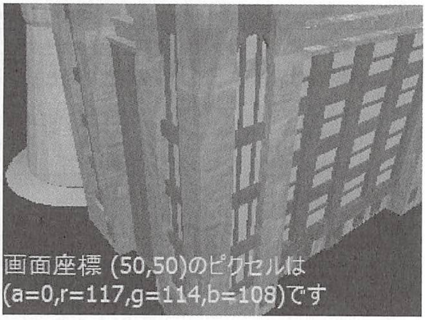
それを行うのがIDirect3DDevice9::GetFrontBufferDataメソッドです。

原理としてはデスクトップ（ビデオメモリ）からシステムメモリのサーフェイスへピクセルをコピー（転送）するもので、それは“非常に”時間が掛かります。また、このメソッドはそれ以上に低速な設計になってもいるようです。リアルタイムパートで使うことは明らかに現実的ではありません。

ただフルスクリーンの場合は、プログラムやドライバーから見ればスクリーンサイズがデスクトップサイズになるので、スクリーンサイズを小さくすればするほど逆累乗的に高速になります。このサンプルのスクリーンサイズをわずか320×240にしているのもそのためです。

さて、フロントバッファの内容を取得する流れを実際のコードで追いましょう。まず、フロントバッファの内容をコピーする受け皿であるサーフェイスを作成します。

▼実行結果



画面座標 (50,50)のピクセルは
(a=0,r=117,g=114,b=108)です

```
//ピクセル情報表示 取得されるピクセルはD3DFMT_A8R8G8B8型
LPDIRECT3DSURFACE9 pCopyFrontBuffer=NULL;
D3DDISPLAYMODE DesktopMode;
pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,&DesktopMode);
pDevice->CreateOffscreenPlainSurface(
    DesktopMode.Width,
    DesktopMode.Height,
    D3DFMT_A8R8G8B8,
    D3DPOOL_SYSTEMMEM,
    &pCopyFrontBuffer,
    NULL);
```

サーフェイスのサイズはスクリーンのサイズと述べておきながら、サイズにデスクトップサイズDesktopMode.Width、DesktopMode.Heightを指定しているように見えますが、これも述べたようにフルスクリーンの場合は、ドライバーから見ればスクリーンサイズすなわちこの場合320×240であり、たとえ実際のデスクトップを2048×1536に設定していようともDesktopMode.Widthは320、DesktopMode.Heightは240の値しか報告しませんので安心してください。

フォーマットはD3DFMT_A8R8G8B8ただ一つのみと決まっています。つまりフロントバッファの値は常にアルファつき32ビットピクセルフォーマットで転送されてくるということです。これはある意味楽ではありますが、32ビットなので速度低下の一端ではあります。サーフェイスはシステムメモリD3DPOOL_SYSTEMMEMに置かなければなりません。

サーフェイスの用意ができたら、ようやくGetFrontBufferDataメソッドを実行して、そのサーフェイスにコピーします。

```
pDevice->GetFrontBufferData(0,pCopyFrontBuffer);
```

この時点で無事サーフェイスに画面ピクセルが格納されているので、あとはサーフェイスをロックして自由にピクセルを調べればいいだけです。

```
D3DLOCKED_RECT Rect;
pCopyFrontBuffer->LockRect(&Rect,NULL,D3DLOCK_READONLY);
BYTE* pPixel=(BYTE*)Rect.pBits;
pPixel+=50*sizeof(DWORD)+50*Rect.Pitch;
//バイトオーダーは逆
BYTE b=*pPixel;
BYTE g=*(pPixel+1);
BYTE r=*(pPixel+2);
BYTE a=*(pPixel+3);
pCopyFrontBuffer->UnlockRect();
SAFE_RELEASE( pCopyFrontBuffer );
```

このサンプルでは座標(50,50)のピクセル一つを調べるようにしています。先頭アドレスから計算して(50,50)のピクセルになるようにしています。（ピッチの重要性についてはTips142を参照してください。）

注意する点は、インテル系CPUを使用しているウィンドウズマシンのメモリオーダーはリトルエンディアンなので、下位バイトが先に現われることです。そのためARGB値（4バイト）を読み取る順序はａｒｇｂではなく真逆のｂｇｒａとなることです。

なお、用事が済んだらサーフェイスはリリースしましょう。こうしないとメモリーリークが発生し、システム全体のパフォーマンスが落ちてしまいます。

以下は、情報の表示部分です。

```
TCHAR  sz[200];
sprintf(sz,"画面座標　(50,50)のピクセルは¥n(a=%d,r=%d,g=%d,b=%d)です",a,r,g,b);
RenderString(sz,0,190);
```

302 デスクトップに表示されているピクセルデータを取得する

デスクトップ上から、指定した座標のピクセルを取得する

上級

サンプルプロジェクト 302 デスクトップのピクセル取得

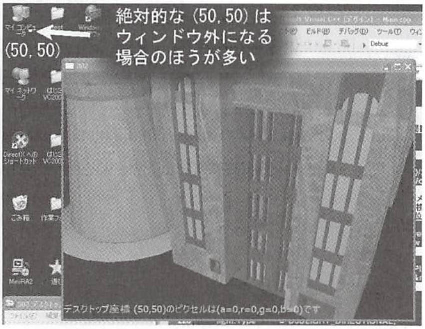
Tips301で述べた理由によりフロントバッファのコピーは非常に低速な処理です。特にウィンドウモードの場合は、ウィンドウのサイズに関り無く、デスクトップがまるごと転送されてしまうので膨大なステップ数になります（Tips301でフルスクリーンの場合、高速になる理由も述べました）。

どれくらい遅いかというと、その時のデスクトップサイズに大きく左右されますが、最高のグラフィックカードを搭載していようとも（CPUはあまり関係ありません）、おそらく10fps出れば良い方でしょうか。ちなみに筆者のマシンはグラフィックスワークステーションと言ってもいい環境ですが、デスクトップサイズが2048×1536なので“3fps”くらいしか出ませんでした。このサンプルをテストする時に限っては、デスクトップサイズを変更しようかと思ったくらいです（アイコンの再配置が面倒なので、3fpsに甘んじましたが）。

原理と手順はTips301と全く同じです。というかコードは99.9%同じです。異なるのは、Direct3Dデバイスをウィドウモードで作成しているという事だけです。

ウィンドウの場合は、フルスクリーンの時のようなスピードアップはありません。そして、ピクセル座標の計算に何も工夫しなければ（本サンプルがそうです）、あくまでデスクトップの座標でしかピクセルを参照できません。つまり、ウィンドウが（0,0）の位置にあれば画面座標（50,50）はウィンドウ内の座標になりますが、ウィンドウをずらしても、絶対的な（50,50）のピクセルを読むことになり、そのピクセルはウィンドウの外である可能性があるということです。

▼実行結果



```
デスクトップ座標 (50,50)のピクセルは(a=0,r=0,g=0,b=0)です
```

303 ウィンドウクライアント画面のピクセルデータを取得する

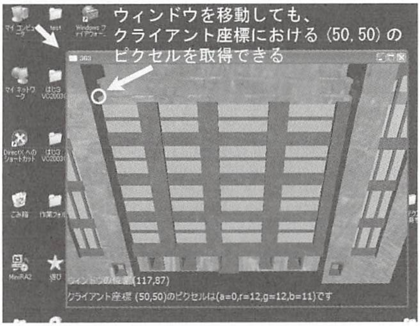
●クライアント座標になるように工夫して、あくまでもシーンのピクセルを取得する　上級

サンプルプロジェクト　303 ウィンドウ内のピクセル取得

直前のTips302の改良版です。このサンプルでは、ピクセルを参照する際のオフセット計算にウィンドウの位置を加えるので、ウィンドウの位置をどう動かそうとも、あくまでクライアント領域（ウィンドウ内）での(50,50)のピクセルを取得することができます。

Tips301,302と異なるのは、クライアント座標をデスクトップ座標に換算する部分だけです。

▼実行結果



ウィンドウを移動しても、クライアント座標における (50, 50) のピクセルを取得できる

ウィンドウの位置(117,87)

クライアント座標 (50,50)のピクセルは(a=0,r=12,g=12,b=11)です

```
//ウィンドウの位置を加味する
WINDOWPLACEMENT  wp;
wp.length=sizeof(WINDOWPLACEMENT);
GetWindowPlacement(hWnd,&wp);
pPixel+=(wp.rcNormalPosition.left+50)*sizeof(DWORD)+(wp.rcNormal
Position.top+50)*Rect.Pitch;
```

One Point

クライアント座標からデスクトップの絶対座標を求めるには、Win32APIのClientToScreen関数を使用してもいいでしょう。

DirectX

500 Tips to Use DirectX Better!

第 4 章

304～313

一歩進んだDirect3Dの極意

4-1　一歩進んだDirect3Dの極意（304～313）

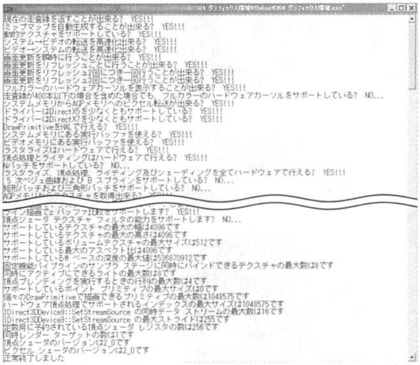
304 ビデオカード及びドライバ関係の能力を取得する

IDirect3DDevice9:: GetDeviceCapsメソッドでデバイスの能力を取得する　初級

サンプルプロジェクト　304　グラフィックス環境

グラフィック関係の能力を取得することは簡単です。Direct3DデバイスのGetDeviceCapsメソッドを実行すれば、D3DCAPS9構造体に情報が入ります。

▼実行結果



```
現在の走査線を返すことが出来る?  YES!!!
ミップマップを自動生成することが出来る?  YES!!!
動的テクスチャをサポートしている?  YES!!!
システム→ビデオの転送を高速化出来る?  YES!!!
ビデオ→システムの転送を高速化出来る?  YES!!!
画面更新を瞬時に行うことが出来る?  YES!!!
画面更新をリフレッシュごとに行うことが出来る?  YES!!!
画面更新をリフレッシュ2回につき一回行うことが出来る?  YES!!!
画面更新をリフレッシュ3回につき一回行うことが出来る?  YES!!!
フルカラーのハードウェアカーソルを表示することが出来る?  YES!!!
走査線が400本以下の場合を含めた場合でも、フルカラーのハードウェアカーソルをサポートしている?  NO...
システムメモリからAGPメモリへのピクセル転送が出来る?  YES!!!
ドライバーはDirectX5を少なくともサポートしている?  YES!!!
ドライバーはDirectX7を少なくともサポートしている?  YES!!!
DrawPrimitiveをHALで行える?  YES!!!
システムメモリにある実行バッファを使える?  YES!!!
ビデオメモリにある実行バッファを使える?  YES!!!
ラスタライズはハードウェアで行える?  YES!!!
頂点処理とライティングはハードウェアで行える?  YES!!!
Nパッチをサポートしている?  NO...
ラスタライズ、頂点処理、ライティング及びシェーディングを全てハードウェアで行える?  YES!!!
5 次ベジェ曲線および B スプラインをサポートしている?  NO...
矩形パッチおよび三角形パッチをサポートしている?  NO...
AGPメモリからテクスチャを取得出来る?  YES!!!
ライン描画でz バッファ比較をサポートします?  YES!!!
頂点シェーダ テクスチャ フィルタの能力をサポートします?  NO...
サポートしているテクスチャの最大の幅は4096です
サポートしているテクスチャの最大の高さは4096です
サポートしているボリュームテクスチャの最大サイズは512です
サポートしている最大のアスペクト比は4096です
サポートしているW ベースの深度の最大値は536870912です
固定機能パイプラインのサンプラ ステージに同時にバインドできるテクスチャの最大数は8です
同時にアクティブにできるライトの最大数は8です
頂点ブレンディングを実行するときの行列の最大数は4です
サポートしているポイント プリミティブの最大サイズは0です
個々のDrawPrimitiveで描画できるプリミティブの最大数は1048575です
ハードウェア頂点処理でサポートされるインデックスの最大サイズは1048575です
IDirect3DDevice9::SetStreamSource の同時データ ストリームの最大数は16です
IDirect3DDevice9::SetStreamSource の最大ストライドは255です
定数用に予約されている頂点シェーダ レジスタの数は256です
同時レンダー ターゲットの数は1です
頂点シェーダのバージョンは2_0です
ピクセル シェーダのバージョンは2_0です
正常終了しました
```

```
D3DCAPS9 Caps9;
pDevice->GetDeviceCaps( &Caps9 );
```

あとは、D3DCAPS9構造体のメンバと知りたい情報の定数との論理積を取ればいいだけです。

```
(Caps9.Caps & D3DCAPS_READ_SCANLINE)!=0 ? printf("¥n 現在の走査線を返すことが出来ます")
              : printf("¥n 現在の走査線を返すことが出来ません");
```

情報量はかなりあり、いちいち上記のようなコードを書いていては見づらくなりますし、なによりコードを打つのが面倒なので、次のようなマクロを作成し、ある程度スッキリさせています。

```
//マクロ
#define YESNO(a,b) printf("¥n%s?",a); (b) != 0 ? printf("  YES!!!") : printf("  NO...");
#define YESNO_INV(a,b) printf("¥n%s?",a); (b) != 0 ? printf("  NO...") : printf("  YES!!!");
```

その能力をサポートしていれば、YES!!!と表示し、そうでなければNO...と表示するようにしています。コードはかなり長いですが、この羅列に過ぎません。

```
//VOID ReserchGraphicsEnv()
//PCのグラフィックス環境を調査（注　マルチモニター等は想定外）
VOID ReserchGraphicsEnv()
{
    D3DCAPS9 Caps9;
    pDevice->GetDeviceCaps( &Caps9 );

    //サポートの有無を表示
    YESNO("現在の走査線を返すことが出来る",Caps9.Caps & D3DCAPS_READ_SCANLINE);
    YESNO("ミップマップを自動生成することが出来る",Caps9.Caps2 & D3DCAPS2_CANAUTOGENMIPMAP);
    YESNO("動的テクスチャをサポートしている",Caps9.Caps2 & D3DCAPS2_DYNAMICTEXTURES);
    YESNO("システム→ビデオの転送を高速化出来る",Caps9.Caps3 & D3DCAPS3_COPY_TO_SYSTEMMEM);
    YESNO("ビデオ→システムの転送を高速化出来る",Caps9.Caps3 & D3DCAPS3_COPY_TO_VIDMEM);
    YESNO("画面更新を瞬時に行うことが出来る",Caps9.PresentationIntervals & D3DPRESENT_INTERVAL_IMMEDIATE);
    YESNO("画面更新をリフレッシュごとに行うことが出来る",Caps9.PresentationIntervals & D3DPRESENT_INTERVAL_ONE);
    YESNO("画面更新をリフレッシュ2回につき一回行うことが出来る",Caps9.PresentationIntervals & D3DPRESENT_INTERVAL_TWO);
    YESNO("画面更新をリフレッシュ3回につき一回行うことが出来る",Caps9.PresentationIntervals & D3DPRESENT_INTERVAL_THREE);
    YESNO("フルカラーのハードウェアカーソルを表示することが出来る",Caps9.CursorCaps & D3DCURSORCAPS_COLOR);
    YESNO("走査線が400本以下の場合を含めた場合でも、フルカラーのハードウェアカーソルをサポートしている",Caps9.CursorCaps & D3DCURSORCAPS_LOWRES);
    YESNO("システムメモリからAGPメモリへのピクセル転送が出来る",Caps9.DevCaps & D3DDEVCAPS_CANBLTSYSTONONLOCAL);
    YESNO("ドライバーはDirectX5を少なくともサポートしている",Caps9.DevCaps & D3DDEVCAPS_DRAWPRIMITIVES2);
    YESNO("ドライバーはDirectX7を少なくともサポートしている",Caps9.DevCaps & D3DDEVCAPS_DRAWPRIMITIVES2EX);
    YESNO("DrawPrimitiveをHALで行える",Caps9.DevCaps & D3DDEVCAPS_DRAWPRIMTLVERTEX);
    YESNO("システムメモリにある実行バッファを使える",Caps9.DevCaps & D3DDEVCAPS_EXECUTESYSTEMMEMORY);
    YESNO("ビデオメモリにある実行バッファを使える",Caps9.DevCaps & D3DDEVCAPS_EXECUTEVIDEOMEMORY);
    YESNO("ラスタライズはハードウェアで行える",Caps9.DevCaps & D3DDEVCAPS_HWRASTERIZATION);
    YESNO("頂点処理とライティングはハードウェアで行える",Caps9.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT);
    YESNO("Nパッチをサポートしている",Caps9.DevCaps & D3DDEVCAPS_NPATCHES);
    YESNO("ラスタライズ、頂点処理、ライティング及びシェーディングを全てハードウェアで行える",Caps9.DevCaps & D3DDEVCAPS_PUREDEVICE);
    YESNO("5次ベジェ曲線およびBスプラインをサポートしている",Caps9.DevCaps & D3DDEVCAPS_QUINTICRTPATCHES);
    YESNO("矩形パッチおよび三角形パッチをサポートしている",Caps9.DevCaps & D3DDEVCAPS_RTPATCHES);
    YESNO("AGPメモリからテクスチャを取得出来る",Caps9.DevCaps & D3DDEVCAPS_TEXTURENONLOCALVIDMEM);
    YESNO("システムメモリからテクスチャを取得出来る",Caps9.DevCaps & D3DDEVCAPS_TEXTURESYSTEMMEMORY);
    YESNO("デバイスメモリからテクスチャを取得出来る",Caps9.DevCaps & D3DDEVCAPS_TEXTUREVIDEOMEMORY);
    YESNO(" 座標変換済みおよびライティング済みの頂点に、システム メモリからのバッファを使える",Caps9.DevCaps & D3DDEVCAPS_TLVERTEXSYSTEMMEMORY);
    YESNO("座標変換済みおよびライティング済みの頂点に、ビデオ メモリからのバッファを使える",Caps9.DevCaps & D3DDEVCAPS_TLVERTEXVIDEOMEMORY);
    YESNO("異方性フィルタリングをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_ANISOTROPY);
    YESNO("色の遠近を正しく補間します",Caps9.RasterCaps & D3DPRASTERCAPS_COLORPERSPECTIVE);
    YESNO("ディザによって色解像度を向上させることができます",Caps9.RasterCaps & D3DPRASTERCAPS_DITHER);
    YESNO("旧来の深度バイアスをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_DEPTHBIAS);
    YESNO("勾配スケール ベースの真の深度バイアスを実行します",Caps9.RasterCaps & D3DPRASTERCAPS_SLOPESCALEDEPTHBIAS);
    YESNO("範囲ベースのフォグをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_FOGRANGE);
    YESNO("ピクセルフォるをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_FOGTABLE);
    YESNO("ライティング処理の最中にフォグ値を計算し、ラスタ化の最中にフォグ値を補間します",Caps9.RasterCaps & D3DPRASTERCAPS_FOGVERTEX);
    YESNO("詳細レベル (LOD) バイアスの調整をサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_MIPMAPLODBIAS);
    YESNO("マルチサンプリングのオンとオフの切り替えをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_MULTISAMPLE_TOGGLE);
    YESNO("シザー テストをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_SCISSORTEST);
    YESNO("wを使った深度バッファリングをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_WFOG);
    YESNO("w ベースのフォグをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_ANISOTROPY);
    YESNO("z ベースのフォグをサポートします",Caps9.RasterCaps & D3DPRASTERCAPS_ZFOG);
    YESNO("z テスト処理を実行できます",Caps9.RasterCaps & D3DPRASTERCAPS_ZTEST);
    YESNO("ハードウェアで隠面消去 (HSR) を実行できます",Caps9.RasterCaps & D3DPRASTERCAPS_ZBUFFERLESSHSR);
    YESNO("グーロー ブレンドされた透明度のアルファ成分をサポートできます",Caps9.ShadeCaps & D3DPSHADECAPS_ALPHAGOURAUDBLEND);
    YESNO("頂点色付きのグーロー シェーディングをサポートできます",Caps9.ShadeCaps & D3DPSHADECAPS_COLORGOURAUDRGB);
    YESNO("グーロー シェーディング モードでフォグをサポートできます",Caps9.ShadeCaps & D3DPSHADECAPS_FOGGOURAUD);
    YESNO("スペキュラ ハイライトのグーロー シェーディングをサポートします",Caps9.ShadeCaps & D3DPSHADECAPS_SPECULARGOURAUDRGB);
    YESNO("テクスチャ ピクセルでアルファがサポートされます",Caps9.TextureCaps & D3DPTEXTURECAPS_ALPHA);
    YESNO("テクスチャ パレットからアルファを描画できます",Caps9.TextureCaps & D3DPTEXTURECAPS_ALPHAPALETTE);
    YESNO("キューブ テクスチャをサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_CUBEMAP);
    YESNO_INV("キューブ テクスチャ マップのディメンジョンが 2 の累乗で指定されている必要がありません",Caps9.TextureCaps & D3DPRASTERCAPS_ZTEST);
    YESNO("ミップマップ化キューブ テクスチャをサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_MIPCUBEMAP);
    YESNO("ミップマップ化テクスチャをサポートします",Caps9.TextureCaps & D3DPRASTERCAPS_ZTEST);
    YESNO("ミップマップ化ボリューム テクスチャをサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_MIPMAP);
    YESNO("ディメンジョンが 2 の累乗でない 2D テクスチャの使用を条件付きでサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_NONPOW2CONDITIONAL);
    YESNO_INV("プログラマブル シェーダおよび固定機能シェーダでの射影バンプ環境参照処理をサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_NOPROJECTEDBUMPENV);
    YESNO("遠近補正テクスチャ処理がサポートされます",Caps9.TextureCaps & D3DPTEXTURECAPS_PERSPECTIVE);
    YESNO_INV("正方形以外のテクスチャえおサポートしている",Caps9.TextureCaps & D3DPTEXTURECAPS_SQUAREONLY);
    YESNO_INV("補間の前にテクスチャ インデックスがテクスチャ サイズに合わせてスケーリングされます",Caps9.TextureCaps & D3DPTEXTURECAPS_TEXREPEATNOTSCALEDBYSIZE);
    YESNO("ボリューム テクスチャをサポートします",Caps9.TextureCaps & D3DPTEXTURECAPS_VOLUMEMAP);
    YESNO_INV("ボリューム テクスチャ マップのディメンジョンが 2 の累乗で指定されている必要が無い",Caps9.TextureCaps & D3DPTEXTURECAPS_VOLUMEMAP_POW2);
    YESNO("[0.0, 1.0] の範囲外の座標に、D3DSAMP_BORDERCOLOR テクスチャ ステージ ス
テートで指定される境界色を設定できます",Caps9.TextureAddressCaps &
D3DPTADDRESSCAPS_BORDER);
    YESNO("テクスチャをアドレスに固定できます",Caps9.TextureAddressCaps &
D3DPTADDRESSCAPS_CLAMP);
    YESNO("テクスチャの u 座標と v 座標のテクスチャ アドレシング モードを分離できます
",Caps9.TextureAddressCaps & D3DPRASTERCAPS_ZTEST);
    YESNO("テクスチャをアドレスに反映させることができます",Caps9.TextureAddressCaps &
D3DPTADDRESSCAPS_MIRROR);
    YESNO("テクスチャ座標の絶対値を取得し 、最大値に固定できる
",Caps9.TextureAddressCaps & D3DPTADDRESSCAPS_MIRRORONCE);
    YESNO("テクスチャをアドレスにラップできます",Caps9.TextureAddressCaps &
D3DPTADDRESSCAPS_WRAP);
    YESNO("ライン描画でアルファ テスト比較をサポートします",Caps9.LineCaps &
D3DLINECAPS_ALPHACMP);
    YESNO("ライン描画でアンチエイリアシングしたラインをサポートします",Caps9.LineCaps &
D3DLINECAPS_ANTIALIAS);
    YESNO("ライン描画で転送元ブレンディングをサポートします",Caps9.LineCaps &
D3DLINECAPS_BLEND);
    YESNO("ライン描画でフォグをサポートします",Caps9.LineCaps & D3DLINECAPS_FOG);
    YESNO("ライン描画でテクスチャ マッピングをサポートします",Caps9.LineCaps &
D3DLINECAPS_TEXTURE);
    YESNO("ライン描画でz バッファ比較をサポートします",Caps9.LineCaps &
D3DLINECAPS_ZTEST);
    YESNO("頂点シェーダ テクスチャ フィルタの能力をサポートします
",Caps9.VertexTextureFilterCaps!=0);

    //値を表示
    printf("¥nサポートしているテクスチャの最大の幅は%dです",Caps9.MaxTextureWidth);
    printf("¥nサポートしているテクスチャの最大の高さは%dです
",Caps9.MaxTextureHeight);
    printf("¥nサポートしているボリュームテクスチャの最大サイズは%dです
",Caps9.MaxVolumeExtent);
    printf("¥nサポートしている最大のアスペクト比は%dです
",Caps9.MaxTextureAspectRatio);
    printf("¥nサポートしているw ベースの深度の最大値は%dです",Caps9.MaxVertexW);
    printf("¥n固定機能パイプラインのサンプラ ステージに同時にバインドできるテクスチャの最大
数は%dです",Caps9.MaxSimultaneousTextures);
    printf("¥n同時にアクティブにできるライトの最大数は%dです",Caps9.MaxActiveLights);
    printf("¥n頂点ブレンディングを実行するときの行列の最大数は%dです
",Caps9.MaxVertexBlendMatrices);
    printf("¥nサポートしているポイント プリミティブの最大サイズは%dです
",Caps9.MaxPointSize);
    printf("¥n個々のDrawPrimitiveで描画できるプリミティブの最大数は%dです
",Caps9.MaxPrimitiveCount);
```

```
    printf("¥nハードウェア頂点処理でサポートされるインデックスの最大サイズは%dです
",Caps9.MaxVertexIndex);
    printf("¥nIDirect3DDevice9::SetStreamSource の同時データ ストリームの最大数は%d
です",Caps9.MaxStreams);
    printf("¥nIDirect3DDevice9::SetStreamSource の最大ストライドは%dです
",Caps9.MaxStreamStride);
    printf("¥n定数用に予約されている頂点シェーダ レジスタの数は%dです
",Caps9.MaxVertexShaderConst);
    printf("¥n同時レンダー ターゲットの数は%dです",Caps9.NumSimultaneousRTs);

#define MAJOR_VER(Number) (((Number)>>8)&0xFF)
#define MINOR_VER(Number) (((Number)>>0)&0xFF)
    printf("¥n頂点シェーダのバージョンは%d_%dです",
MAJOR_VER(Caps9.VertexShaderVersion),MINOR_VER(Caps9.VertexShaderVersion));
    printf("¥nピクセル シェーダのバージョンは%d_%dです",
MAJOR_VER(Caps9.PixelShaderVersion),MINOR_VER(Caps9.PixelShaderVersion));
}
```

305 自分の環境でのシェーダーの情報を取得する

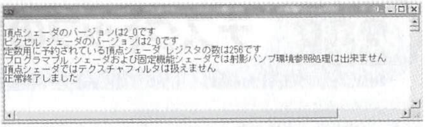
IDirect3DDevice9:: GetDeviceCapsメソッドで利用可能なシェーダーの情報を取得する　初級

サンプルプロジェクト　305 シェーダーの情報

シェーダーの情報もIDirect3DDevice9::GetDeviceCapsメソッドで取得できます。

シェーダーのバージョンは1個のDWORD値で表されているので、これを2つの値に分解する必要があります。メジャーバージョン番号とマイナーバージョン番号です。それぞれの値を取り出すためのD3DSHADER_VERSION_MAJORとD3DSHADER_VERSION_MINORマクロが用意されているので、ここではそれを使用しています。

▼実行結果



```
頂点シェーダのバージョンは2_0です
ピクセル シェーダのバージョンは2_0です
定数用に予約されている頂点シェーダ レジスタの数は256です
プログラマブル シェーダおよび固定機能シェーダでは射影バンプ環境参照処理は出来ません
頂点シェーダではテクスチャフィルタは扱えません
正常終了しました
```

```
VOID ShowShaderCaps()
{
    D3DCAPS9 Caps9;
    pDevice->GetDeviceCaps( &Caps9 );
    printf("¥n頂点シェーダのバージョンは%d_%dです",
          D3DSHADER_VERSION_MAJOR(Caps9.VertexShaderVersion),
          D3DSHADER_VERSION_MINOR(Caps9.VertexShaderVersion));

    printf("¥nピクセル シェーダのバージョンは%d_%dです",
          D3DSHADER_VERSION_MAJOR(Caps9.PixelShaderVersion),
          D3DSHADER_VERSION_MINOR(Caps9.PixelShaderVersion));

    printf("¥n定数用に予約されている頂点シェーダ レジスタの数は%dです
",Caps9.MaxVertexShaderConst);

    if( (Caps9.TextureCaps & D3DPTEXTURECAPS_NOPROJECTEDBUMPENV) !=0)
    {
          printf("¥nプログラマブル シェーダおよび固定機能シェーダでの射影バンプ環境参照処
理をサポートします");
    }
    else
    {
          printf("¥nプログラマブル シェーダおよび固定機能シェーダでは射影バンプ環境参照処
理は出来ません");
    }
    if( Caps9.VertexTextureFilterCaps!=0)
    {
          printf("¥n頂点シェーダでテクスチャ フィルタの能力をサポートします");
    }
    else
    {
          printf("¥n頂点シェーダではテクスチャフィルタは扱えません");
    }
}
```

306 デスクトップのサイズ、フォーマットを取得する

IDirect3D9:: GetAdapterDisplayModeメソッドでデスクトップ情報を取得する 初級

参考プロジェクト 301、302、303

デスクトップのサイズ、フォーマット等を取得するには次のようにします。pD3dはDirect3Dオブジェクトポインターです。

```
D3DDISPLAYMODE DesktopMode;
pD3d->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,&DesktopMode);
```

これでDesktopModeにはデスクトップの情報が入ります。

`DesktopMode.Width` デスクトップの横サイズ（単位はピクセル）

`DesktopMode.Height` デスクトップの縦サイズ（単位はピクセル）

`DesktopMode.Format` デスクトップのフォーマット

`DesktopMode.RefreshRate` 現在のデスクトップで設定しているモニターのリフレッシュレート

307 サーフェイスを画像ファイルにする

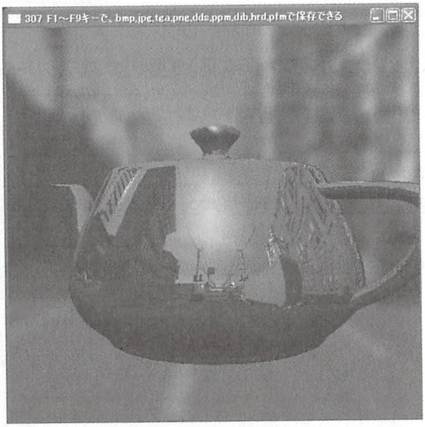
サーフェイスのスナップショットをファイルとして記録する

初級

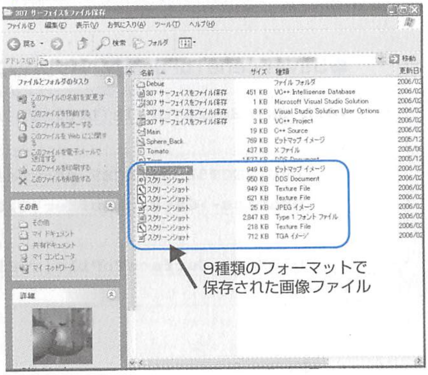
サンプルプロジェクト　307 サーフェイスをファイル保存

バックバッファのスナップショットをファイルに落とす（つまりスクリーンショット）には、D3DXSaveSurfaceToFile関数を使用します

▼実行時画面


307 F1～F9キーで、bmp,jpg,tga,png,dds,ppm,dib,hrd,pfmで保存できる

▼画像ファイルが出来上がる


307 サーフェイスをファイル保存
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
ヘルプ(H)
戻る
検索
フォルダ
アドレス(D)
移動
ファイルとフォルダのタスク
このファイルの名前を変更する
このファイルを移動する
このファイルをコピーする
このファイルを Web に公開する
このファイルを電子メールで送信する
このファイルを印刷する
このファイルを削除する
その他
その他
マイ ドキュメント
共有ドキュメント
マイ コンピュータ
マイ ネットワーク
詳細

名前	サイズ	種類	更新日
Debug		ファイル フォルダ	2006/02
307 サーフェイスをファイル保存	451 KB	VC++ Intellisense Database	2006/02
307 サーフェイスをファイル保存	1 KB	Microsoft Visual Studio Solution	2006/02
307 サーフェイスをファイル保存	8 KB	Visual Studio Solution User Options	2006/02
307 サーフェイスをファイル保存	3 KB	VC++ Project	2006/02
Main	19 KB	C++ Source	2006/02
Sphere_Back	769 KB	ビットマップ イメージ	2005/12
Tomato	437 KB	X ファイル	2005/09
スクリーンショット	949 KB	ビットマップ イメージ	2006/02
スクリーンショット	950 KB	DDS Document	2006/02
スクリーンショット	949 KB	Texture File	2006/02
スクリーンショット	621 KB	Texture File	2006/02
スクリーンショット	25 KB	JPEG イメージ	2006/02
スクリーンショット	2,847 KB	Type 1 フォント ファイル	2006/02
スクリーンショット	218 KB	Texture File	2006/02
スクリーンショット	712 KB	TGA イメージ	2006/02

9種類のフォーマットで保存された画像ファイル

●ウィンドウプロシージャー内でキー入力により、異なるフォーマットを指定してSaveSceneToFIle関数をコール

```
case VK_F1:
    SaveSceneToFIle("スクリーンショット.bmp",D3DXIFF_BMP);
    break;
case VK_F2:
    SaveSceneToFIle("スクリーンショット.jpg",D3DXIFF_JPG);
```

```
        break;
    case VK_F3:
        SaveSceneToFIle("スクリーンショット.tga",D3DXIFF_TGA);
        break;
    case VK_F4:
        SaveSceneToFIle("スクリーンショット.png",D3DXIFF_PNG);
        break;
    case VK_F5:
        SaveSceneToFIle("スクリーンショット.dds",D3DXIFF_DDS);
        break;
    case VK_F6:
        SaveSceneToFIle("スクリーンショット.ppm",D3DXIFF_PPM);
        break;
    case VK_F7:
        SaveSceneToFIle("スクリーンショット.dib",D3DXIFF_DIB);
        break;
    case VK_F8:
        SaveSceneToFIle("スクリーンショット.hdr",D3DXIFF_HDR);
        break;
    case VK_F9:
        SaveSceneToFIle("スクリーンショット.pfm",D3DXIFF_PFM);
        break;
```

●SaveSceneToFIle関数。バックバッファのスナップショットをファイルに落とす関数(つまりスクリーンショット)

```
VOID SaveSceneToFIle(LPCTSTR szFileName,D3DXIMAGE_FILEFORMAT Format)
{
    LPDIRECT3DSURFACE9 pSourceSurface=NULL;
    pDevice->GetBackBuffer(0,0,D3DBACKBUFFER_TYPE_MONO,&pSourceSurface);

    D3DXSaveSurfaceToFile(szFileName,Format,pSourceSurface,NULL,NULL);
}
```

One Point

Tips307の注意点

この関数で作成されるビットマップは32ビットビットマップなので、開けないソフトがあります。筆者のフォトショップ5.0では開けませんでした。むしろ、OS付属の簡易ビューアー、あるいはフォルダのプレビューのほうが確実に見ることができました。

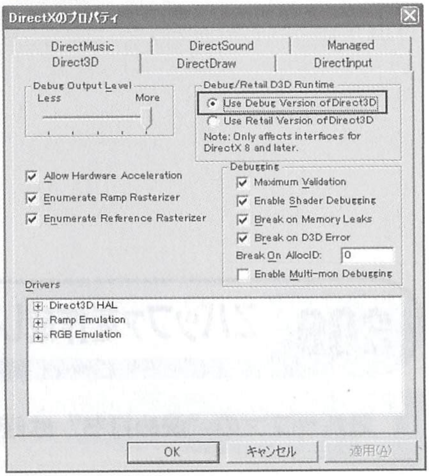
308 デバッグ時の“ウォッチ”でインターフェイスのメンバを表示するには?

●#define D3D_DEBUG_INFOを追加してデバッグ情報を表示する　初級

Direct3Dプログラムをデバッガーでデバッグしている時、ウォッチウィンドウでインターフェイスが次のようにしか表示されずに不便だと感じたことがないでしょうか？Direct3Dインターフェイスは、次の操作をすることによりウォッチでメンバの詳細を表示することが出来ます。

その前にまず、右の図のようにコントロールパネル→DirectXアイコンをクリック→DirectXダイアログを出して、Direct3Dランタイムがデバッグバージョンに設定していることを確認し、もしリテールバージョンを設定しているなら、デバッグバージョンに変更してください。

▼実行結果



Debug Output Level
Less　More

Debug/Retail D3D Runtime
● Use Debug Version of Direct3D
○ Use Retail Version of Direct3D
Note: Only affects interfaces for DirectX 8 and later.

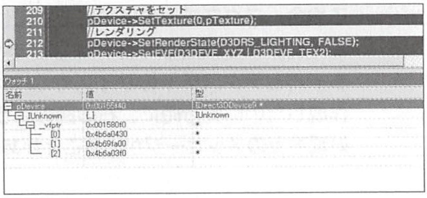
☑ Allow Hardware Acceleration
☑ Enumerate Ramp Rasterizer
☑ Enumerate Reference Rasterizer

Debugging
☑ Maximum Validation
☑ Enable Shader Debugging
☑ Break on Memory Leaks
☑ Break on D3D Error
Break On AllocID: 0
☐ Enable Multi-mon Debugging

Drivers
Direct3D HAL
Ramp Emulation
RGB Emulation

次に、ソースコード上で、#include <d3dx9.h>の前に次のディファイン文を追加します。

▼実行結果



```
//テクスチャをセット
pDevice->SetTexture(0,pTexture);
//レンダリング
pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX2);
```

名前	値	型
pDevice	[illegible]	IDirect3DDevice9 *
IUnknown	{...}	IUnknown
__vfptr	0x001580f0	*
[0]	0x4b6a0430	*
[1]	0x4b69fa00	*
[2]	0x4b6a03f0	*

```
#define  D3D_DEBUG_INFO
```

例えば、最初次のようになっている場合、

```
#include  <windows.h>
#include  <d3dx9.h>
```

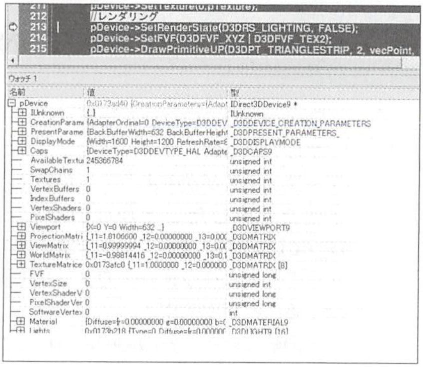
次のように変更するだけです。

```
#define  D3D_DEBUG_INFO
#include  <windows.h>
```

```
#include <d3dx9.h>
```

これだけです。これでインターフェイスはウォッチウィンドウで次のように表示されます。

▼実行結果



```
pDevice->SetTexture(0,pTexture);
//レンダリング
pDevice->SetRenderState(D3DRS_LIGHTING, FALSE);
pDevice->SetFVF(D3DFVF_XYZ | D3DFVF_TEX2);
pDevice->DrawPrimitiveUP(D3DPT_TRIANGLESTRIP, 2, vecPoint,
```

```
ウォッチ 1
名前                値                                          型
⊟ pDevice          0x0173ad40 {CreationParameters={Adapt      IDirect3DDevice9 *
  ⊞ IUnknown        {...}                                       IUnknown
  ⊞ CreationParam   {AdapterOrdinal=0 DeviceType=D3DDEV          _D3DDEVICE_CREATION_PARAMETERS
  ⊞ PresentParame   {BackBufferWidth=632 BackBufferHeight        _D3DPRESENT_PARAMETERS_
  ⊞ DisplayMode     {Width=1600 Height=1200 RefreshRate=6        _D3DDISPLAYMODE
  ⊞ Caps            {DeviceType=D3DDEVTYPE_HAL Adapte            _D3DCAPS9
    AvailableTextu  245366784                                    unsigned int
    SwapChains      1                                            unsigned int
    Textures        1                                            unsigned int
    VertexBuffers   0                                            unsigned int
    IndexBuffers    0                                            unsigned int
    VertexShaders   0                                            unsigned int
    PixelShaders    0                                            unsigned int
  ⊞ Viewport        {X=0 Y=0 Width=632 ...}                      _D3DVIEWPORT9
  ⊞ ProjectionMatri {_11=1.8106600 _12=0.00000000 _13=0.00(      _D3DMATRIX
  ⊞ ViewMatrix      {_11=0.99999994 _12=0.00000000 _13=0.0(      _D3DMATRIX
  ⊞ WorldMatrix     {_11=-0.98814416 _12=0.00000000 _13=0.1      _D3DMATRIX
  ⊞ TextureMatrice  0x0173afc0 {_11=1.0000000 _12=0.000000       _D3DMATRIX [8]
    FVF             0                                            unsigned long
    VertexSize      0                                            unsigned int
    VertexShaderV   0                                            unsigned long
    PixelShaderVer  0                                            unsigned long
    SoftwareVertex  0                                            int
  ⊞ Material        {Diffuse={r=0.00000000 g=0.00000000 b=(      _D3DMATERIAL9
  ⊞ Lights          0x0173b218 {Type=0 Diffuse={r=0.00000(       _D3DLIGHT9 [16]
```

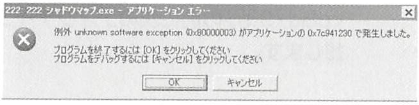
309 Zバッファを作成しただけなのに実行時例外エラーがでる

●Zバッファの“決め打ち”は厳禁！

中級

テクスチャをレンダリングターゲットにする場合、テクスチャのほかにそれに付随したZバッファも作成しなければなりませんが、ちゃんとテクスチャのサイズでZバッファを作成したのに実行時に、右の図のような例外処理違反のメッセージが出ることがあります。

▼実行結果



```
222: 222 シャドウマップ.exe - アプリケーション エラー
例外 unknown software exception (0x80000003) がアプリケーションの 0x7c941230 で発生しました。

プログラムを終了するには [OK] をクリックしてください
プログラムをデバッグするには [キャンセル] をクリックしてください
```

この原因として最も可能性が高いのは、Zバッファのサイズを“決め打ち”で作成してしまっていることです。

例えば次のように、

```
pDevice->CreateDepthStencilSurface(WINDOW_WIDTH,WINDOW_HEIGHT,
            D3DFMT_D16,D3DMULTISAMPLE_NONE, 0, TRUE, &pTexZBuffer, NULL);
```

Direct3DデバイスやテクスチャをWINDOW_WIDTH、WINDOW_HEIGHTで作成しているからと思って安心してZバッファもそれと同じサイズにするのは危険です。テクスチャは作成した時に渡したサイズで作成されているとは限らないからです。たとえば800×600のテクスチャを作成したつもりでも実際に作成されるのは1024×1024だったりします。

ですので、そのテクスチャの情報を取得し、その情報からサイズを指定することが最も確実です。

```
D3DSURFACE_DESC desc;
pDepthTexture->GetLevelDesc(0,&desc);
pDevice->CreateDepthStencilSurface(desc.Width,desc.Height,
            D3DFMT_D16,D3DMULTISAMPLE_NONE, 0, TRUE, &pTexZBuffer, NULL);
```

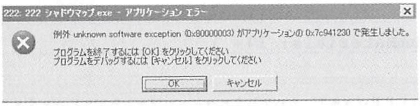
310 シェーダーで原因不明の実行時例外エラーがでる

シェーダーによる原因不明のクラッシュ

中級

プログラマブルシェーダーが原因で例外違反メッセージが出ていることが分かっていても、シェーダーコードのどこにも原因が無いように思うことがあるかもしれません。

▼実行結果



例外 unknown software exception (0x80000003) がアプリケーションの 0x7c941230 で発生しました。

プログラムを終了するには [OK] をクリックしてください
プログラムをデバッグするには [キャンセル] をクリックしてください

そんな時は、次の事項を確認してください。

シェーダーコードで、どこにも使用されないメンバを定義しても問題はありませんが、そのメンバに代入を行うだけで、クラッシュすることがあります。

次のように頂点シェーダーの入出力構造体を定義しているとして、float4 Colは実際には、無意味であり無くても良いメンバだとします。

```
struct VS_OUT
{
    float4 Pos : POSITION;
    float4 Col : COLOR0;
    float2 TC : TEXCOORD0;
};
```

そのような無駄なメンバを定義しているだけなら問題ありませんが、そのメンバに1回でも代入処理を行うと、それだけでクラッシュすることがあります。

```
VS_OUT VS_Blur(float4 Position : POSITION, float2 TC : TEXCOORD0)
{
    VS_OUT Out = (VS_OUT)0;
    Out.Pos = Position;
    Out.TC=TC;
    Out.Col = float4(1,1,1,1);←実際、このColメンバはピクセルシェーダーで使用していな
い。全くの無駄

    return Out;
}
```

代入したのならどこかで使わなければいけません。というよりも、無駄なメンバは削除したほうがいいですね。

311 シェーダーでテクスチャ、マテリアル等の変更が反映されない

●パスの中での設定変更

中級

シェーダーでは、一つのパスの中でなんらかの変更（SetValueによるマテリアル変更、SetTextureによるテクスチャの再セットなど）を加えたときは、ID3DXEffect::CommitChangesメソッドを実行します。これを行わないと、変更が反映されません。

実際のアプリ側のコードは、例えば次のようになります。

```
// レンダリング
 {
     for( DWORD i=0; i<pThing->dwNumMaterials; i++ )
     {
             pEffect->SetValue("Diffuse",&pThing->pMeshMaterials[i].
Diffuse,sizeof(FLOAT)*4 );
             pEffect->CommitChanges();
             pThing->pMesh->DrawSubset( i );
     }
 }
```

312 メッシュがカメラに近くなると“くり抜かれる”

●カメラ付近でポリゴンが欠ける減少に対処する

初級

D3DXMatrixPerspectiveFovLH（またはD3DXMatrixPerspectiveFovRH）の第4引数を低めに設定すると解消されます。

第4引数はニアクリップ面までの距離を意味し、ポリゴンとカメラ（視点）の距離がそれ以下の距離にあるとレンダリングされず、結果としてそのポリゴン部分だけくり抜かれる形になります。

One Point

逆に遠くのポリゴンが欠ける場合は、第5引数のファークリップ面までの距離を高くすることで解決する場合があります。

313 重なっているポリゴンに汚い“しましま”が入る

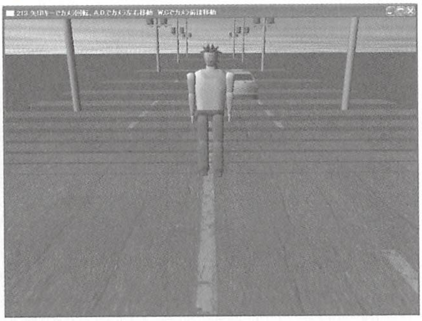
●Zファイティングが原因でポリゴンが見えたり見えなかったりする現象に対処する　初級

狭い間隔で重なっているポリゴン同士が交互にちらちら見え隠れすることは無いでしょうか？
浮動小数点数の計算精度からくる問題であり、ある瞬間は表側と判断されても別の瞬間では裏側と判断されることに起因します。

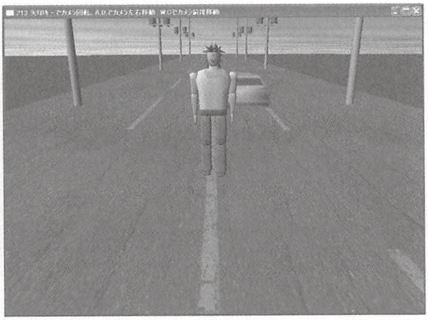
これはZファイティング(Z fighting)（Z値同士が戦ってしまう）と呼ばれる現象です。簡単な方法としてはD3DXMatrixPerspectiveFovLH（またはD3DXMatrixPerspectiveFovRH）の第4引数をZファイティングを起こしているポリゴン同士の隙間より高く設定することです。

第4引数はニアクリップ面までの距離を意味するものですが、Zファイティングの発生を抑制する作用もあります。ニアクリップ面の値がポリゴン間の隙間より相対的に小さいとZファイティングが起こりやすくなってしまいます。

▼ニアクリップ　0.01f


213 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Sでカメラ前後移動

▼ニアクリップ　0.1f


213 矢印キーでカメラ回転、A,Dでカメラ左右移動 W,Sでカメラ前後移動

第2部
DirectSound の極意

DirectX
Visual C++ 2002/2003/2005対応
▶逆引き大全◀
500の極意

DirectX

500 Tips to Use DirectX Better!

第 5 章

314～323

WAVファイルの極意

5-1　WAVファイルを知る（314～323）

314 WAVファイルとは？

●マイクロソフトが策定したサウンドファイルの規格

初級

WAVファイルは、マイクロソフトが策定したサウンドファイルの規格であり、ご存知のようにウィンドウズにおける標準のサウンドファイルで、拡張子が.wavのファイルです。

より詳しく言うと、WAVファイルはRIFFファイル（Tips 317）の一種であり、圧縮フォーマットを含めると130以上の種類があります。（130種類と言っても、もちろん全て拡張子は.wavです）。

DirectSoundがサポートしているサウンドフォーマットは、RIFFのWAVE形式だけです。つまり、mp3フォーマットやwmaフォーマット形式には対応していません。また、同じWAVEフォーマットでも圧縮WAVEフォーマットには対応していません。非圧縮のWAVEフォーマット、非圧縮のWAVファイルが唯一DirectSoundで再生できるフォーマットです。

315 WAVファイルの構造

●最初にフォーマット部分があり、残りがデータ部分

初級

WAVファイルの中身は、まず最初にフォーマット部分があり、残りがデータ部分というように分かれています。

フォーマット部分には、サンプリング周波数、ビット深度、データサイズ等が記載されています。データ部分は、文字通り波形データの羅列です。

フォーマット部分はサウンドの長さに依存しないので、そのサイズは常に数十バイト程度です。

データ部分は、（当然ですが）サウンドの長さに比例してどんどん大きくなります。

One Point

詳しいデータの格納形式については、Tips317「RIFFとは？」を参照してください。

また、データ（波形データ）そのものの詳細についてはTips318「PCMとは？」を参照してください。

316 WAVファイルの種類

130以上の形式がある

初級

一口にWAVファイルと言っても、非常に多くの種類があります。筆者は現在130以上の形式を認識していますが、もっとあるかもしれません。ただ、そのほとんどは圧縮形式のWAVファイルであり、DirectSoundには直接関係ありません。DirectSoundは非圧縮（リニアPCM）のWAVファイルしか再生しません。

なお、圧縮形式とはmp3やwma等のことではなく、あくまでもWAVファイルの圧縮タイプを言いますので、混同しないようにしてください。圧縮WAVファイルは、構造が同じつまりRIFFでありつつ、波形データになんらかの圧縮をするものです。それに対しmp3やwmaはフォーマットそのものが異なり、構造的に全く異なります。

317 RIFFとは？

動画関係のAVIフォーマットの仕様

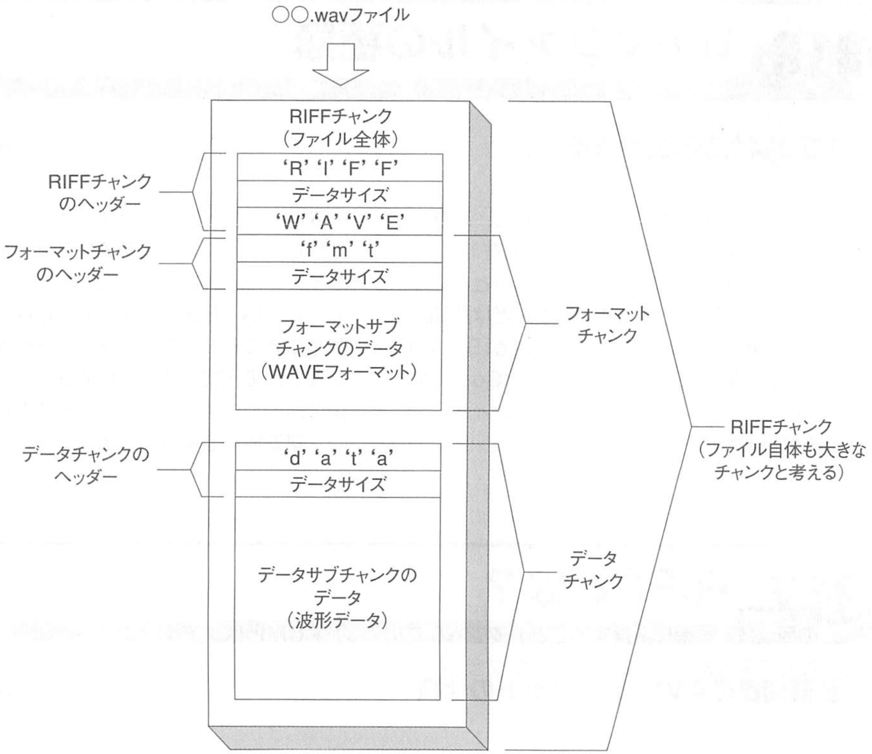
中級

RIFFのフルネームはResource Interchange File Formatであり、WAVEフォーマットはもちろん、動画関係のAVIフォーマットの仕様でもあります。

RIFFはマイクロソフトとIBMが共同で作成したフォーマット仕様ですが、ベースはEA（Electronic Arts)というゲームメーカーが策定したIFF（Interchange File Format）です。

ちなみにAppleは、IFFをベースにAIFFというオーディオファイルフォーマット仕様を作成し、また、MMA（MIDI Manufacturers Association）はIFFをベースにMIDIファイルの規格SMF（Standard MIDI File Format）を策定していたりします。

IFF、RIFF（RIFFはデータの並び順が逆のIFFです）の原理は明瞭かつ単純で、タグを付けたデータブロック（これをチャンクと言います）を論理単位としてファイルに格納するというもので、タグベースフォーマットとも呼ばれています。


○○.wavファイル
RIFFチャンク
（ファイル全体）
'R' 'I' 'F' 'F'
データサイズ
'W' 'A' 'V' 'E'
'f' 'm' 't'
データサイズ
フォーマットサブ
チャンクのデータ
（WAVEフォーマット）
'd' 'a' 't' 'a'
データサイズ
データサブチャンクの
データ
（波形データ）
RIFFチャンク
のヘッダー
フォーマットチャンク
のヘッダー
データチャンクの
ヘッダー
フォーマット
チャンク
データ
チャンク
RIFFチャンク
（ファイル自体も大きな
チャンクと考える）

チャンクは、まず一番先頭にIDを持ちます（これをタグとも呼びます）。IDは単なるアスキーコード4文字です。したがってID自体のサイズは4バイトです。

IDの次には、そのチャンクが持つデータのサイズが来ます。そのチャンク全部のサイズではなく、あくまでもそのチャンクのデータ部分のサイズであることに注意してください。データのサイズは32ビット整数で表現されるので、データサイズ（を現すもの）自体は4バイトです。

その次にようやく、そのチャンクがもつデータが来ます。

チャンクは入れ子状になっているので、すこしややこしいかもしれませんが、ファイル全体をRIFFチャンクという大きなチャンクと考えると、次のようになります。

RIFFチャンクのデータ部分は、フォーマットチャンクとデータ（波形）チャンクです。

フォーマットチャンクのデータ部分は、周波数、ビット深度等です。

データ（波形）チャンクのデータ部分は、波形データです。

それぞれのサイズは

RIFF　ID=4
RIFF　データサイズ=4
フォームID=4

フォーマットチャンクID=4
フォーマットチャンクサイズ=4
フォーマットチャンクデータサイズ=sizoef(WAVEFORMATEX)　または

sizoef(WAVEFORMATEXTENSIBLE)

データチャンクID=4
デーチャンクサイズ=（波形の全サイズ）

318 PCMとは？

●パルス符号変調の一つ

中級

PCMのフルネームはPulse Code Modulationで、日本語では「パルス符号変調」と呼ばれます。

PCMは、“変調方式”の一つです。

では、変調（モジュレーション：Modulation）とは何なのか？

広義の意味では、「ある周波数・振幅を別の周波数・振幅に変換すること」ですが、ここでの変調とは「元となるアナログ音を、デジタルデータに変換する」とでも覚えてください。

PCMは、DirectSound（というよりもコンピューターサウンド）ばかりではなく、一般のオーディオ機器、テレビ、ラジオ、電話、言うなれば音をデジタル信号に、逆に、デジタル信号をアナログ音にする場面全てに使用されている変調・復調方式（復調とは変調した信号を音に戻すこと、ここではデジタルからアナログへの逆変換です）です。

まとめると

変調とは、アナログ音をデジタルデータにすること。

PCMは変調の1つの方法ということ。

したがって、

PCMとは、アナログ音→デジタルデータ変換のことです。

PCMが変調を行うとき、流れは次の3段階になります。

①標本化
②量子化
③符号化

それぞれの詳細については、Tips320～322を参照してください。

Column

変調方式いろいろ

昔、PCにはFM音源というFMによる変調・復調方式であるサウンドハードウェアが搭載されていましたが、FMの解像度はかなり低く、結果的に出力される音も貧弱なものでした。FM音源が登場するよりも前、パソコン黎明期には、PSG音源というものもありました。これは、音声波形を矩形により近似するという最も大雑把な近似で、曲っぽい音を出すのにはトリッキーなコーディングが必要でした。ただし「ピコピコ音」を出すだけでいいような状況であれば、メモリ的なコストパフォーマンスが最も優れた変調方式ではあります。これが当時の非力なハードで採用されていた理由でもあります。またこの理由により現在でも安価なおもちゃ等で「ピコピコ音」にはPSG音源が使用されるということをどこかで聞いたことがあります。

319 リニアPCMとは？

波形データを圧縮していないタイプのもの

中級

WAVファイルの中で波形データを圧縮していないタイプのものを、リニアPCMであるWAVファイルと言います。

DirectSoundは8ビットあるいは16ビットのリニアPCMのみサポートしています。

リニアPCMで重要なフォーマットは、次の3つだけです。普段我々が運用するWAVファイルのほとんどがこの3つのどれかだと思ってもいいでしょう。

WAVE_FORMAT_PCM
WAVE_FORMAT_EXTENSIBLE
WAVE_FORMAT_IEEE_FLOAT

WAVE_FORMAT_PCM型のWAVEファイルが基本ですが、マルチチャンネルWAVファイル関係のコーディングにはWAVE_FORMAT_EXTENSIBLE型のWAVEファイルを用いることになります。WAVE_FORMAT_IEEE_FLOAT型とは、サンプル（波形）を32ビットFLOAT型、つまり、浮動小数点数で表現するタイプですが、この型のWAVEファイルにについては考えなくて構いません。なぜなら、基本的にDirectSoundは8ビットあるいは16ビットのリニアPCMのみサポートしているからです。

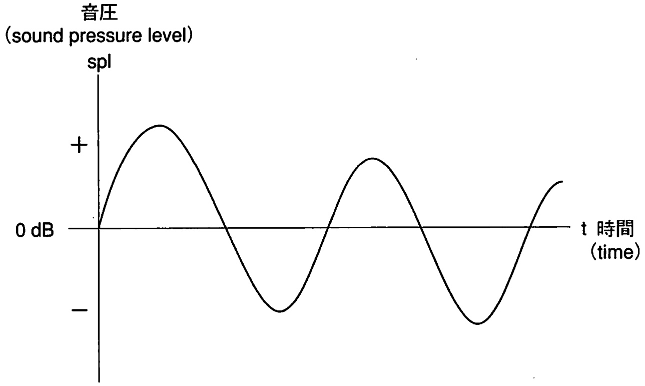
なお、この3つ識別子は、RIFFチャンクのフォーマット内で記述され、フォーマットを判別する際に使用するものでもあります。

320 標本化の原理

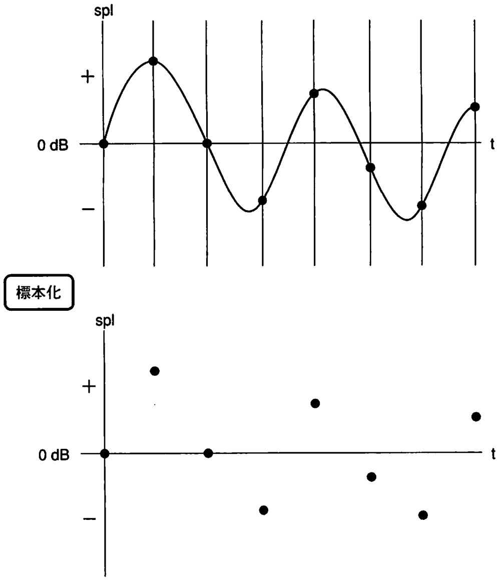
標本化の原理を知る

中級

標本化（Sampling：サンプリング)はPCM変換の過程で一番最初に行われる変換です。その目的は、時間ごとの波形のスナップショットを抽出することです。次ページの図は1秒間の波形とします。


音圧
(sound pressure level)
spl
+
0 dB
−
t 時間
(time)

音は、横軸に時間、縦軸に音圧をとるグラフ上では波形として表現されます。最初の状態は変調していない状態です。


spl
+
0 dB
−
t
標本化
spl
+
0 dB
−
t

これを、縦横に分割していきます。それぞれの分割数は、視認しやすいように実際よりもかなり少ない数にして解説します。

まず横に8分割します。この音圧の瞬間値の１本１本が音のサンプルです。１秒間に8個のサンプルを抽出するのでサンプリング周波数8ヘルツです。超超低周波なので聞くことはできないでしょう。

サンプルはアナログ量のスナップショットなので、この時点でのサンプルは実数値になります。実数値では具合が悪いので、次の量子化により整数値にします。

321 量子化の原理

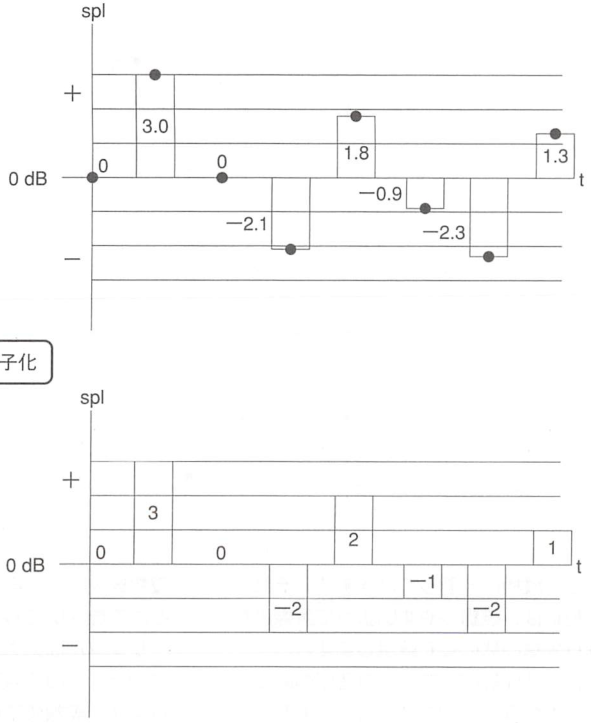
●サンプルを意図したビット数で表現できるようにする

中級

量子化（Quantization：クオンティゼイション）の目的は実数値であるサンプルを意図したビット数で表現できるようにすることです。通常はサンプルの小数部分を量子化特性により四捨五入し、整数値にすることですが、32ビットフロートでは小数点数も表現できるので、その場合は整数にする必要はありません。いずれにしても、量子化の結果として、サンプルを意図したビット数の範囲で表せるようになります。

ここでは、縦に3分割しているので、3ビットあればいいように見えますが、グラフから分るようにサンプルは振幅0を境にして、プラスとマイナスがありますので、プラスマイナスを表すビットが一つ必要です。したがってこの場合、量子化ビット数は4ビット必要です。量子化ビット数はビット深度とも呼ばれます。

この時点で、デジタルデータにするための準備ができたことになります。



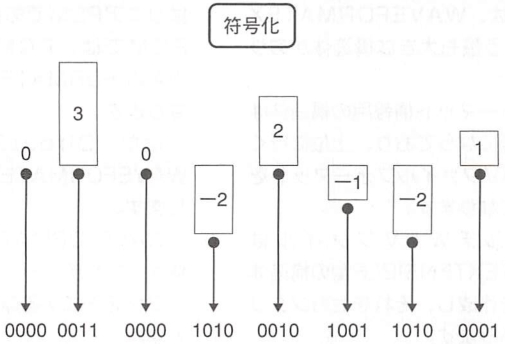
標本化から同じ波形で話を進めているので、この音は8ヘルツ、4ビットのサウンドデータと言えます。実際はもっと大きな周波数、ビット深度になりますが、本質は同じです。

322 符号化の原理

量子化後のサンプルを0と1に分解すること

中級

符号化（encoding：エンコーディング）の目的は、量子化後のサンプルを0と1に分解することです。PCにおけるファイル的データとしてなら、量子化の段階でビット表現に変換されているので、既に目的は達成されています。この符号化は考えなくていいかもしれません。ただ、PCM変換は符号化までを含めた変換なのであえて掲載しました。



Column

CDはなぜ44.1キロヘルツなのか？

ナイキスト周波数を基にしたシャノンの標本化定理というものがあり、それは「標本から元の信号を再現するには、元信号の最高周波数の2倍以上の周波数で標本化すれ良い」というものです。PCMに適用すると、「原音の2倍以上の周波数で変調すれば、原音に復調できる」ことなります。　音楽CDが44.1キロヘルツであるのは、人間が認識できる音の帯域が20ヘルツ～20キロヘルツであると言われていたため、その2倍に若干の余裕を持たせた値であるからです。ただ、最近では人間はもっと高い周波数を認識できるということが分かったため、量子化ビット数24ビットでサンプリング周波数が192キロヘルツという超高音質なDVD-Audioが登場したりしています。

323 サウンドフォーマット構造体を把握する

●PCMWAVEFORMAT、WAVEFORMATEX、WAVEFORMATEXTENSIBLEの関係を把握する　中級

WAVEファイルのフォーマットチャンクが最低限持つ情報を格納する構造体はPCMWAVEFORMATです。

PCMWAVEFORMATよりもっと下位にWAVEFORMATというさらに小さな構造体がありますが、実際には使用されず、実質的に基本となるフォーマットチャンクの最低情報はPCMWAVEFORMAT型からです。

その上位にWAVEFORMATEX型があり、さらにその上には、WAVEFORMATEXTENSIBLEという最も大きな構造体があります。

このようにフォーマット情報用の構造体は綺麗に入れ子状態になっており、上位に行くほど、複雑なWAVファイルフォーマットを表現できるようになります。

例えば、マルチWAVファイルはWAVEFORMATEXTENSIBLE型の構造体でフォーマットを作成し、それをセカンダリバッファに詰め込みます。

PCMWAVEFORMAT構造体は、リニアPCMのWAVEファイルに収録されているフォーマットに1体1に対応する完全な情報を持ちます。

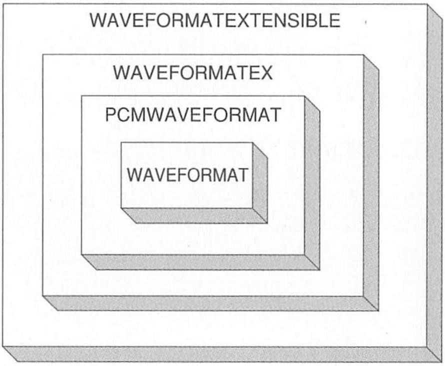
DirectSoundはリニアPCMのみ再生するものなので、PCMWAVEFORMAT構造体さえあれば、通常のリニアPCM（ステレオまでのリニアPCM）のフォーマットは表現できます。

WAVEFORMATEX型は、PCMWAVEFORMAT型に「cbSize」というメンバを付け加えただけです。このメンバはリニアPCMであればゼロなので、リニアPCMでは、PCMWAVEFORMATもWAVEFORMATEXも実質的には同じ値になります。

ただ、DirectSoundバッファには、WAVEFORMATEX型でフォーマットを渡します。

WAVEFORMATEXTENSIBLE型の構造体は、マルチチャンネルのDirectSoundバッファを作成する際に使用する特殊な構造体です。

通常、ステレオまでなら（2チャンネルまでなら）、WAVEFORMATEX型しか使用しません。



WAVEFORMAT

```
typedef struct {
WORD wFormatTag;
WORD nChannels;
DWORD nSamplesPerSec;
DWORD nAvgBytesPerSec;
WORD nBlockAlign;
} WAVEFORMAT;
```

PCMWAVEFORMAT

```
typedef struct {
WAVEFORMAT wf;
WORD wBitsPerSample;
} PCMWAVEFORMAT;
```

＝

WAVEFORMAT

```
typedef struct {
WORD wFormatTag;
WORD nChannels;
DWORD nSamplesPerSec;
DWORD nAvgBytesPerSec;
WORD nBlockAlign;
} WAVEFORMAT;
```

＋

```
WORD wBitsPerSample;
```

WAVEFORMATEX

```
typedef struct {
WORD wFormatTag;
WORD nChannels;
DWORD nSamplesPerSec;
DWORD nAvgBytesPerSec;
WORD nBlockAlign;
WORD wBitsPerSample;
WORD cbSize;
} WAVEFORMATEX;
```

＝

PCMWAVEFORMAT

```
typedef struct {
WAVEFORMAT wf;
WORD wBitsPerSample;
} PCMWAVEFORMAT;
```

＋

```
WORD cbSize;
```

WAVEFORMATEXTENSIBLE

```
typedef struct {
WAVEFORMATEX Format;
union {
WORD wValidBitsPerSample;
WORD wSamplesPerBlock;
WORD wReserved;
} Samples;
DWORD dwChannelMask;
GUID SubFormat;
} WAVEFORMATEXTENSIBLE,
*PWAVEFORMATEXTENSIBLE;
```

＝

WAVEFORMATEX

```
typedef struct {
WORD wFormatTag;
WORD nChannels;
DWORD nSamplesPerSec;
DWORD nAvgBytesPerSec;
WORD nBlockAlign;
WORD wBitsPerSample;
WORD cbSize;
} WAVEFORMATEX;
```

＋

```
union {
WORD wValidBitsPerSample;
WORD wSamplesPerBlock;
WORD wReserved;
} Samples;
DWORD dwChannelMask;
GUID SubFormat;
```

第 6 章

324～334

初期化の極意

6-1　DirectSoundの初期化（324～325）
6-2　WAVデータ読み込み（326～334）

324 DirectSoundオブジェクトを作成する

全ての始まりDirectSoundオブジェクトを作成する

初級

DirectSoundをコーディングするには、DirectSoundのヘッダーをインクルードし、ライブラリをロードしなくてはならないので、次のコードがソースの最初にでも必要になります。

```
#include  <Dsound.h>
#pragma  comment(lib,  "dsound.lib")
#pragma  comment(lib,  "dxguid.lib")
```

DirectSoundオブジェクトのポインターを用意します。

```
LPDIRECTSOUND8  g_pDSound;
```

実質的なコードはたった一行です。

```
DirectSoundCreate8(NULL,&g_pDSound,NULL);
```

325 協調レベルを設定する

協調レベルを設定する

初級

協調レベルとは他のアプリケーションのサウンドに対し、どれだけ優先させるかという意味があります。

ここでhWndはウィンドウハンドル、g_pDSoundは適正に作成されたDirectSoundオブジェクトとします。

協調レベルの設定は次のようになります。

```
g_pDSound->SetCooperativeLevel(hWnd,DSSCL_EXCLUSIVE  );
```

第2引数で協調レベルを指定します。次の2つのレベルがあります。

DSSCL_EXCLUSIVEまたはDSSCL_PRIORITY
他のアプリに対し優先。

DSSCL_NORMAL
他のアプリと同等。

One Point

最近のDirectSound、また、特にWindowsXP環境（WDMドライバモデル）ではフラグはあまり意味を持ちません。

326　WAVファイルを開く

Windows Multimedia APIを利用してWAVファイルを操作する　中級

サンプルプロジェクト　326-334　WAV読み込みと再生

WAVファイルと言っても通常の“ファイル”ですから、皆さんも良く書くであろう、

FILE ＊fp=fopen()

というコードで開いて、中身を参照、変更することができます。

ただ、チャンクへファイルポインタを移動する操作、つまりファイル内の走査には専用のAPIが用意されていて、それを使用すればコードがスッキリ短くなります。

専用のAPIとはWindows Multimedia SDKというSDKのAPIです。特別な設定は要りません。ライブラリファイルをロードすれば、そのまま使用できます。

```
#pragma comment(lib, "winmm.lib")
```

ファイルを開くには次のようなコードになります。

```
HMMIO hMmio=NULL;
hMmio = mmioOpen( “何とか.wav” , NULL, MMIO_ALLOCBUF | MMIO_READ );
```

このあとの処理（チャンクの読み込み）については、Tips327以降で解説します。

327 WAVファイル内を走査してチャンクを見つける

●mmioDescend関数でファイルポインタを意図したチャンク先頭にセットする

中級

サンプルプロジェクト　326-334 WAV読み込みと再生

チャンクへの移動を想定したAPIがあります。Windows Multimedia APIのmmioDescend関数です。

チャンク走査コードの一般的な形は次のようになります。

```
MMCKINFO   ckInfo;
ckInfo.ckid  =  mmioFOURCC('f',  'm',  't',  ' ');
mmioDescend(  hMmio,  &ckInfo,  &riffckInfo,  MMIO_FINDCHUNK  );
```

MMCKINFO構造体を用意し、そのckidメンバに意図したチャンクのFOURCC値を代入れます（FOURCCについてはTips330を参照）。

この例ではfmtチャンク（フォーマットチャンク）にファイルポインタを進めようとしています。

mmioDescend関数はファイルポインタをそのチャンク先頭にセットする機能と、ファイルのチャンクを読み込むという機能もあります。そのためアプリケーションで用意したckInfo構造体には、mmioDescendした時点でファイル内のチャンク情報が格納されます。

hMmioはファイルを開いた時に返されるファイルハンドルです（FILE＊ fpのfpのようなものです）。

328 WAVファイルかどうかを判別する

●RIFFチャンクヘッダーを調べて判別する

中級

サンプルプロジェクト　326-334 WAV読み込みと再生

そのファイルがWAVEファイルかどうか、RIFFチャンクのヘッダーを調べることにより判断できます。

というかそもそもRIFFファイルかどうか（RIFFチャンクが存在するかどうか）を調べることでも判断できます。RIFFファイルでなければWAVファイルであるはずがありません。Tips327で解説した手法でRIFFチャンクにファイルポインタをセットして読み込めばいいわけです。

```
MMCKINFO  ckInfo;
ckInfo.ckid  =  mmioFOURCC('R',  'I',  'F',  'F');
mmioDescend(  hMmio,  &ckInfo,  &riffckInfo,  MMIO_FINDCHUNK  );
```

これでももちろん良いのですが、RIFFチャンクは唯一特別なチャンクで必ずファイル先頭にあることが分かっている総元締め的なチャンクですから次のように省略したコードを書くこともでき、サンプルではこの書き方をしています。

```
MMCKINFO  riffckInfo;
mmioDescend(  hMmio,  &riffckInfo,  NULL,  0  );
```

あとは、読み込んだRIFFチャンクヘッダーのメンバを調べればいいわけです。

```
if(  (riffckInfo.ckid  !=  mmioFOURCC('R',  'I',  'F',  'F'))  ||  (riffckInfo.
fccType  !=  mmioFOURCC('W',  'A',  'V',  'E')  )  )
{
    MessageBox(0,"これはWAVファイルではありません","  lol  ",MB_OK);
    return  E_FAIL;
}
```

329 リニアPCMかどうかを判別する

●フォーマットチャンクのデータ部分を読み込み、リニアPCMかどうか判断する 中級

サンプルプロジェクト 326-334 WAV読み込みと再生

mmioDescend関数はチャンクヘッダーを読み込むのに対し、チャンクのデータ部分を含めて、チャンク全体を読み込むにはmmioRead関数を使用します。

少々ややこしいですが、フォーマットチャンクを丸ごとを読み込み、リニアPCMかどうか判断します。

PCMWAVEFORMAT構造体は、フォーマットチャンクそのものを表す構造体です(RIFFを策定したのがマイクロソフト自身なので、ファイル内のチャンクと同じ内容の構造体があるのもうなづけます)。ですので、ファイル内のチャンクデータをそのまま構造体に放り込めばいいので楽です。

```
PCMWAVEFORMAT  pcmWaveFormat;
//フォーマットを読み込む
mmioRead(  hMmio,   (HPSTR)  &pcmWaveFormat,sizeof(pcmWaveFormat));
//リニアPCMで、かつ、マルチチャンネルWAVは想定外
if(  pcmWaveFormat.wf.wFormatTag   ==  WAVE_FORMAT_PCM  )
{

}
else
{
    MessageBox(0,"本サンプルは標準のリニアPCMフォーマットを想定しています","",MB_OK);
    return  E_FAIL;
}
```

リニアPCMの場合、フォーマットのwFormatTagメンバが必ずWAVE_FORMAT_PCMになります。逆に言うとそうなっていないファイルはリニアPCMではないことになります。

330 FOURCCとは？

単なる32ビット整数

初級

サンプルプロジェクト　326-334 WAV読み込みと再生

コード中で、mmioFOURCC('R', 'I', 'F', 'F')とかmmioFOURCC('W', 'A', 'V', 'E')などとFOURCC値を作成しています。

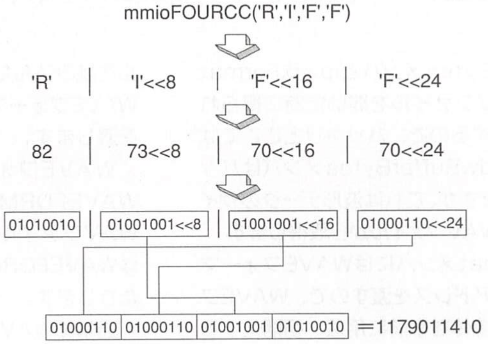
FOURCC値は単なる32ビット整数です。コード上ではチャンクのタグをFOURCC値で扱います。

mmioFOURCCは関数ではなくマクロです。なんならavifmt.hヘッダーファイルのマクロ定義を見てみてください。単純なものであることが分かるでしょう。

```
#define  mmioFOURCC(  ch0,  ch1,  ch2,  ch3  )                              ¥
          (  (DWORD)(BYTE)(ch0)  |  (  (DWORD)(BYTE)(ch1)  <<  8  )  |  ¥
          (  (DWORD)(BYTE)(ch2)  <<  16  )  |  (  (DWORD)(BYTE)(ch3)  <<  24
)  )
```

マクロの仕組みは簡単で、4文字の各8ビット32ビットをビットフィールドに配置して32ビット整数にします。マクロを展開して、例えばRIFFの32ビット値を作る行程を

示すと次のようになります。


mmioFOURCC('R','I','F','F')
'R'
'I'<<8
'F'<<16
'F'<<24
82
73<<8
70<<16
70<<24
01010010
01001001<<8
01001001<<16
01000110<<24
01000110
01000110
01001001
01010010
＝1179011410

Rのアスキーコードは82、Iは73、Fは70、それを2進数にしたものをビットシフトして連結したときのビットパターンを32ビット値と見立てた値がFORUCC値です。ビットシフトしているのは、RIFFがリトルエンディアンなので、各バイトの並び順を逆にしなくてはならないからです。

331 セカンダリーバッファを作成する

セカンダリバッファの作成は様々

中級

サンプルプロジェクト　326-334　WAV読み込みと再生

作成そのものは、1行で済み、次のようになります（g_pDSoundはDirectSoundオブジェクトポインターです）。

```
g_pDSound->CreateSoundBuffer(  &dsbd,  &g_pDSBuffer,  NULL);
```

面倒なのは第1引数です。第1引数はDSBUFFERDESC型の構造体のアドレスを渡します。その構造体の初期化は様々なバリエーションがあり決まった形で書けませんが、基本的なパターンを示します。

```
DSBUFFERDESC  dsbd;
ZeroMemory(  &dsbd,  sizeof(DSBUFFERDESC)  );
dsbd.dwSize  =  sizeof(DSBUFFERDESC);
dsbd.dwFlags  =NULL
dsbd.dwBufferBytes  =(バッファのサイズ)
dsbd.lpwfxFormat  =  (WAVEフォーマット構造体のアドレス）;
```

dwBufferBytesメンバとlpwfxFormatメンバはWAVファイルを開いた時に得られる情報を基にするので、ハッキリとここでは書けません。dwBufferBytesメンバはバッファのサイズですが、これは波形データのサイズを意味し、WAVファイルから取得します。

lpwfxFormatメンバにはWAVEフォーマット構造体のアドレスを渡すので、WAVEフォーマット構造体を事前に用意・初期化することも必要になります。通常はWAVファイルを読み込んだ際に得られたフォーマットでWAVEフォーマット構造体を初期化し、それを渡します。

WAVEフォーマット構造体は、大抵はWAVEFORMATEX型でしょうが、マルチWAVファイルからバッファを作成する場合はWAVEFORMATEXTENSIBLE型になったりします。

通常のWAVファイルであれば、次のようになります。

```
WAVEFORMATEX  pwfex;
(pwfexを初期化。と言っても通常はWAVファイル内のフォーマット部分をpwfexにコピーするだけ)。
dsbd.lpwfxFormat=&pwfex;
```

dwFlagsをNULLにしています。これでも単純に音を出すことはできますが、目的に応じて様々な指定ができます。例えば、プログラム内でボリュームを操作できるようにしたい場合は、次のように指定します。

```
dsbd.dwFlags  =  DSBCAPS_CTRLVOLUME;
```

ボリューム操作に加えて、サウンドの周波数も動的に変更したい場合（ユーザーの入力によりエンジン音を変化させるとか）は、次のように論理和で指定します。

```
dsbd.dwFlags  =  DSBCAPS_CTRLVOLUME  |  DSBCAPS_CTRLFREQUENCY;
```

このように、追加したい機能はどんどん論理和で追加していけばいいのです。フラグの種類と意味については、SDKヘルプドキュメントを参照してください。

332 セカンダリバッファにWAVフォーマットを格納する

●DSBUFFERDESCのlpwfxFormatメンバにフォーマット構造体のアドレスを渡す 中級

サンプルプロジェクト 326-334 WAV読み込みと再生

pcmWaveFormatには、WAVファイルのフォーマットが格納されます。

```
PCMWAVEFORMAT  pcmWaveFormat;
mmioRead(  hMmio,  (HPSTR)  &pcmWaveFormat,sizeof(pcmWaveFormat))
```

以下の時点で、pwfexにはフォーマット情報が全て格納されています。

```
WAVEFORMATEX＊  pwfex;
pwfex  =  (WAVEFORMATEX＊)new  CHAR[  sizeof(WAVEFORMATEX)  ];
memcpy(  pwfex,  &pcmWaveFormat,  sizeof(pcmWaveFormat)  );←PCMWAVEFORMAT型
とWAVEFORMATEX型はほとんど同じ。cbSizeメンバの有無だけ。
pwfex->cbSize  =  0;←PCMWAVEFORMAT型には無く、WAVEFORMATEX独自のメンバは手動で設定。
```

あとは、バッファ作成時の第1引数の構造体に、pwfexを渡せばバッファ作成時に自動的にコピーされます。

```
DSBUFFERDESC  dsbd;
ZeroMemory(  &dsbd,  sizeof(DSBUFFERDESC)  );
dsbd.dwSize  =  sizeof(DSBUFFERDESC);
dsbd.dwFlags  =NULL
dsbd.dwBufferBytes  =(バッファのサイズ)
dsbd.lpwfxFormat  =  pwfex;

g_pDSound->CreateSoundBuffer(  &dsbd,  &g_pDSBuffer,  NULL);
```

333 セカンダリーバッファに波形データを書き込む

●バッファをロックして、バイト単位でWAVデータをコピーするz

初級

サンプルプロジェクト　326-334 WAV読み込みと再生

サンプルコードから、エラー処理部分を省いたコードを掲載します。通常のfopen、freadを使用しているので分かりやすいと思います。流れとしては、まず、ファイルから読み込んだ波形データを一時的に保管するために、一時的なメモリ領域を用意します。そのメモリ領域にファイルから波形データを読みこみます。そのメモリ領域の内容をDirectSoundバッファに普通にコピーします。

```
//波形データを書き込む準備として、セカンダリーバッファをロックする
VOID＊  pBuffer  =NULL;
DWORD  dwBufferSize  =  0;
g_pDSBuffer->Lock(  0,  dwWavSize,&pBuffer,  &dwBufferSize,NULL,  NULL,  0);
//バッファに音データを書き込む
FILE＊  fp=fopen(szFileName,"rb");
fseek(fp,riffckInfo.dwDataOffset  +  sizeof(FOURCC),SEEK_SET);
BYTE＊  pWavData=new  BYTE[  dwSize  ];
fread(pWavData,1,dwSize,fp);//まずは、一時的な場所(pWavData)に波形データを読み込み
for(  DWORD  i  =  0;  i  <  dwSize;  i++  )
{
        ＊((BYTE＊)pBuffer+i)  =  ＊((BYTE＊)  pWavData+i);  //一時的な場所の波形デ
ータをセカンダリバッファにコピーする
}
fclose(fp);
delete  pWavData;//一時的な波形データを開放する
g_pDSBuffer->Unlock(  pBuffer,  dwBufferSize,  NULL,  0  );
```

気をつける点としては、DirectSoundバッファの先頭ではなく（先頭部分はフォーマット情報の場所）、データ部分に波形を突っ込むことです。

ファイルポインタをDirectSoundバッファの波形保存場所にセットしているコードは、次の部分です。

```
fseek(fp,riffckInfo.dwDataOffset  +  sizeof(FOURCC),SEEK_SET);
```

334 WAVファイル読み込み全コード

エラー処理部分を省いたWAVファイル読み込みコードの全体を見る 中級

ンプルコード片 334 WAV読み込みコード

これは、Tips326から333までに使用したサンプルコードからエラーコードを省いたコードです。したがって、本質的には全く同じコードです。何かの役に立つかもしれないと思い、一応掲載しておきます。

```
HRESULT  InitSound(HWND  hWnd,LPSTR  szFileName)
{
    HMMIO  hMmio=NULL;//Windowsマルチメディア APIのハンドル(Windowsマルチメディア APIはWAVファイル関係の操作用のAPIでありDirectXではない)
    DWORD  dwWavSize  =  0;//WAVファイル内  WAVデータのサイズ（WAVファイルはWAVデータで占められているので、ほぼファイルサイズと同一）
    WAVEFORMATEX＊  pwfex;//WAVのフォーマット  例）16ビット、44100Hz、ステレオなど
    MMCKINFO  ckInfo;//  チャンク情報
    MMCKINFO  riffckInfo;//  最上部チャンク（RIFFチャンク）保存用
    PCMWAVEFORMAT  pcmWaveFormat;
          //まず一番最初にすることはグローバル関数DirectSoundCreate8によりDirectSoundオブジェクトを生成すること
      DirectSoundCreate8(NULL,&g_pDSound,NULL);
      //  協調レベルの設定
      g_pDSound->SetCooperativeLevel(hWnd,DSSCL_EXCLUSIVE  );
      //WAVファイル内のヘッダー情報（音データ以外）の確認と読み込み
     hMmio  =  mmioOpen(  szFileName,  NULL,  MMIO_ALLOCBUF  |  MMIO_READ  );
      //ファイルポインタをRIFFチャンクの先頭にセットする
      mmioDescend(  hMmio,  &riffckInfo,  NULL,  0  );
      //  ファイルポインタを'f'  'm'  't'  '  '  チャンクにセットする
      ckInfo.ckid  =  mmioFOURCC('f',  'm',  't',  '  ');
      mmioDescend(  hMmio,  &ckInfo,  &riffckInfo,  MMIO_FINDCHUNK  );
      //フォーマットを読み込む
      mmioRead(  hMmio,  (HPSTR)  &pcmWaveFormat,sizeof(pcmWaveFormat));
      pwfex  =  (WAVEFORMATEX＊)new  CHAR[  sizeof(WAVEFORMATEX)  ];
      memcpy(  pwfex,  &pcmWaveFormat,  sizeof(pcmWaveFormat)  );
      pwfex->cbSize  =  0;
      mmioAscend(  hMmio,  &ckInfo,  0  );
      //  WAVファイル内の音データの読み込み
      ckInfo.ckid  =  mmioFOURCC('d',  'a',  't',  'a');
     mmioDescend( hMmio, &ckInfo, &riffckInfo, MMIO_FINDCHUNK );//データ
チャンクにセット
   dwWavSize = ckInfo.cksize;
     // DirectSoundセカンダリーバッファ作成
   DSBUFFERDESC dsbd;
     ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
     dsbd.dwSize = sizeof(DSBUFFERDESC);
     dsbd.dwFlags = NULL;
     dsbd.dwBufferBytes = dwWavSize;
     dsbd.guid3DAlgorithm = DS3DALG_DEFAULT;
     dsbd.lpwfxFormat = pwfex;
     g_pDSound->CreateSoundBuffer( &dsbd, &g_pDSBuffer, NULL );
     //波形データを書き込む準備として、セカンダリーバッファをロックする
     VOID＊ pBuffer =NULL;
     DWORD dwBufferSize = 0;
     g_pDSBuffer->Lock( 0, dwWavSize,&pBuffer, &dwBufferSize,NULL, NULL, 0);

     //バッファに音データを書き込む
     FILE＊ fp=fopen(szFileName,"rb");
     fseek(fp,riffckInfo.dwDataOffset + sizeof(FOURCC),SEEK_SET);
     BYTE＊ pWavData=new BYTE[ dwBufferSize ];
     fread(pWavData,1,dwBufferSize,fp);//まずは、一時的な場所(pWavData)に波形デー
タを読み込み
     for( DWORD i = 0; i < dwBufferSize; i++ )
     {
        ＊((BYTE＊)pBuffer+i) = ＊((BYTE＊) pWavData+i); ////まずは、一時的
な場所の波形データをセカンダリバッファにコピーする
     }
     fclose(fp);
     delete pWavData;//一時的な波形データを開放する
     g_pDSBuffer->Unlock( pBuffer, dwBufferSize, NULL, 0 );

     return S_OK;
}
```

DirectX

500 Tips to Use DirectX Better!

第 7 章

335～417

再生の極意

7-1　基本再生（335～343）
7-2　ストリーミング再生（344～356）
7-3　エフェクト再生（357～392）
7-4　3D再生（393～404）
7-5　マルチチャンネル再生（405～417）

335 セカンダリーバッファを再生する

●IDirectSoundBuffer::Playメソッドで再生する

初級

参考サンプルプロジェクト　338　静的バッファ再生

セカンダリバッファに読み込んだWAVデータは、IDirectSoundBuffer::Playメソッドで再生できます。

サンプルでは次のようにして再生を行っています。

```
g_pDSBuffer->Play(NULL,0,NULL);
```

もちろんg_pDSBufferはIDirectSoundBufferインターフェイスのポインターで、適正に初期化され、かつ、WAVデータをロードしているものとします。

Playメソッドの引数の意味は以下のとおりです。

第1引数は必ずNULLにします。
第2引数はプライオリティですが、あまり意味がありません。0にします。
第３引数は、再生方法のオプションで、DSBPLAY_LOOPINGを指定するとサウンドを繰り返し再生するようになります。

336 セカンダリーバッファを停止する

●IDirectSoundBuffer::Stopメソッドで再生する

初級

セカンダリーバッファーを停止するということは、つまり今鳴っているサウンドを停止するということで、IDirectSoundBuffer::Stopメソッドで行います。
g_pDSBuffer->Stop();

もちろんg_pDSBufferはIDirectSoundBufferインターフェイスのポインターです。
なお、引数はありません。

このメソッドを実行すると、再生カーソル（詳しくはTips347　「ストリーミングの仕組み」を参照）はその位置に留まり、次回に再生したときにはその位置から再生されます。

要するに、次回Playメソッドを実行したときには、サウンドの頭からではなく、止めた位置から再生されるということです。

337 いま再生中かどうか判断する

●IDirectSoundBuffer::GetStatusメソッドで再生状態を取得する

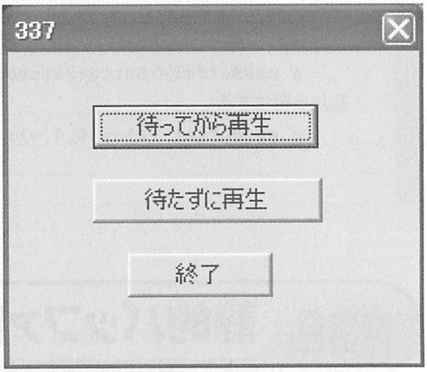
初級

参考サンプルプロジェクト 337 待機再生と即時再生

サンプルでは、再生中かどうかの判断をすることはもちろん、加えて再生カーソルを頭にもっていくことにより、即時再生と待機再生を実現しています。即時再生とはそのサウンドが再生中であってもすぐにまた頭から再生するもので、待機再生とは再生中であれば、再生が終わるまで待って、終了した時点で次の再生をするものです。なにもしなければ通常は待機再生になります。

今再生中であるかどうかは、IDirectSoundBuffer:: GetStatusメソッドを実行し、得られたDWORD値を調べることにより判断します。コードは次のようになります。

▼実行結果



```
DWORD  dwStatus;
g_pDSBuffer->GetStatus(&dwStatus);
```

dwStatusとDSBSTATUS_PLAYINGの論理積をとり、非ゼロであれば再生中ということになります。

```
if((dwStatus  &  DSBSTATUS_PLAYING)  !=0)
```

サンプルのコードは次のようになっています。

●ダイアログプロシージャー内

```
case  IDC_BUTTON1:
    PlaySound(TRUE);//待ってから次の再生
return  TRUE;
case  IDC_BUTTON3:
    PlaySound(FALSE);//待たずにすぐ再生
```

●PlaySound関数

```
VOID  PlaySound(BOOL  boWait)
{
    DWORD  dwStatus;
    g_pDSBuffer->GetStatus(&dwStatus);

    if(  boWait==TRUE  &&  (dwStatus  &  DSBSTATUS_PLAYING)  !=0)←再生中であれ
ばリターンする（待機再生の場合）
    {
          return;
    }
    g_pDSBuffer->SetCurrentPosition(0);←即時再生であれば、再生カーソルを頭に戻し、最
初から再生する
    g_pDSBuffer->Play(0,0,0);
}
```

338 静的バッファによる基本的な再生を行う

●DirectSoundバッファに全てのWAVデータを格納した再生を行う

中級

サンプルプロジェクト　338　静的バッファ再生

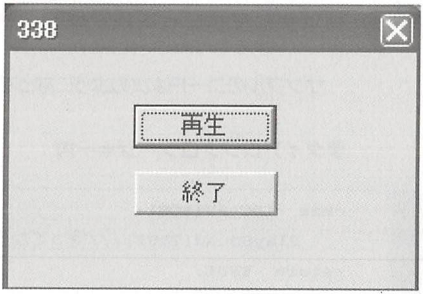
DirectSoundにはWAVファイルのローダーオブジェクト等が用意されていないので、WAVファイルを手作業でロードしなければならず、どうしてもコードが長くなってしまいますが、このサンプルはこれ以上簡略化できないというところまで簡略化しています。

流れは次のようになります。

①まずは、DirectSoundオブジェクトを作成し、
②協調レベルを設定し、
③WAVファイルをDirectSoundバッファにロードします。
④ここまでが完了すれば、あとは再生をするだけです。

①〜③、全てInitSound関数が行っています。

▼実行結果



```
HRESULT  InitSound(HWND  hWnd,LPSTR  szFileName)
{
    HMMIO   hMmio=NULL;//WindowsマルチメディアAPIのハンドル(WindowsマルチメディアAPI
はWAVファイル関係の操作用のAPIでありDirectXではない)
    DWORD  dwWavSize  =  0;//WAVファイル内  WAVデータのサイズ（WAVファイルはWAVデータ
で占められているので、ほぼファイルサイズと同一)
    WAVEFORMATEX＊  pwfex;//WAVのフォーマット  例）16ビット、44100Hz、ステレオなど
    MMCKINFO  ckInfo;//  チャンク情報
    MMCKINFO  riffckInfo;//  最上部チャンク（RIFFチャンク）保存用
    PCMWAVEFORMAT  pcmWaveFormat;
        //まず一番最初にすることはグローバル関数DirectSoundCreate8によりDirectSoundオ
ブジェクトを生成すること
      DirectSoundCreate8(NULL,&g_pDSound,NULL);
      //  協調レベルの設定
      g_pDSound->SetCooperativeLevel(hWnd,DSSCL_EXCLUSIVE  );
      //WAVファイル内のヘッダー情報（音データ以外）の確認と読み込み
      hMmio  =  mmioOpen(  szFileName,  NULL,  MMIO_ALLOCBUF  |  MMIO_READ  );
      //ファイルポインタをRIFFチャンクの先頭にセットする
      mmioDescend(  hMmio,  &riffckInfo,  NULL,  0  );
      //  ファイルポインタを'f'  'm'  't'  '  '  チャンクにセットする
      ckInfo.ckid  =  mmioFOURCC('f',  'm',  't',  '  ');
      mmioDescend(  hMmio,  &ckInfo,  &riffckInfo,  MMIO_FINDCHUNK  );
      //フォーマットを読み込む
      mmioRead(  hMmio,  (HPSTR)  &pcmWaveFormat,sizeof(pcmWaveFormat));
      pwfex  =  (WAVEFORMATEX＊)new  CHAR[  sizeof(WAVEFORMATEX)  ];
      memcpy(  pwfex,  &pcmWaveFormat,  sizeof(pcmWaveFormat)  );
      pwfex->cbSize  =  0;
      mmioAscend(  hMmio,  &ckInfo,  0  );
      //  WAVファイル内の音データの読み込み
      ckInfo.ckid  =  mmioFOURCC('d',  'a',  't',  'a');
      mmioDescend(  hMmio,  &ckInfo,  &riffckInfo,  MMIO_FINDCHUNK  );//データ
チャンクにセット
    dwWavSize  =  ckInfo.cksize;
      //  DirectSoundセカンダリーバッファ作成
    DSBUFFERDESC  dsbd;
      ZeroMemory(  &dsbd,  sizeof(DSBUFFERDESC)  );
      dsbd.dwSize  =  sizeof(DSBUFFERDESC);
      dsbd.dwFlags  =  NULL;
      dsbd.dwBufferBytes  =  dwWavSize;
      dsbd.guid3DAlgorithm  =  DS3DALG_DEFAULT;
      dsbd.lpwfxFormat  =  pwfex;
      g_pDSound->CreateSoundBuffer(  &dsbd,  &g_pDSBuffer,  NULL  );
      //波形データを書き込む準備として、セカンダリーバッファをロックする
      VOID＊  pBuffer  =NULL;
    DWORD  dwBufferSize  =  0;
    g_pDSBuffer->Lock(  0,  dwWavSize,&pBuffer,  &dwBufferSize,NULL,  NULL,
0);

    //バッファに音データを書き込む
    FILE＊  fp=fopen(szFileName,"rb");
    fseek(fp,riffckInfo.dwDataOffset  +  sizeof(FOURCC),SEEK_SET);
    BYTE＊  pWavData=new  BYTE[  dwBufferSize  ];
    fread(pWavData,1,dwBufferSize,fp);//まずは、一時的な場所(pWavData)に波形デー
タを読み込み
    for(  DWORD  i  =  0;  i  <  dwBufferSize;  i++  )
    {
        ＊((BYTE＊)pBuffer+i)  =  ＊((BYTE＊)  pWavData+i);  ////まずは、一時的
な場所の波形データをセカンダリバッファにコピーする
    }
    fclose(fp);
    delete  pWavData;//一時的な波形データを開放する
    g_pDSBuffer->Unlock(  pBuffer,  dwBufferSize,  NULL,  0  );

    return  S_OK;
}
```

再生はプロシージャー内で行っています。

```
g_pDSBuffer->Play(NULL,0,NULL);
```

339 サウンドをパンさせる

●IDirectSoundBuffer:: SetPanメソッドによりパンさせる 初級

サンプルプロジェクト 339 パン

パンとは、左右のスピーカーの相対ボリュームを変更することです。

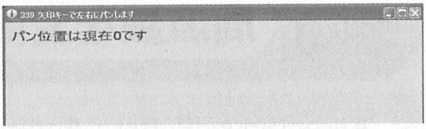
例えば左にパンさせるということは左スピーカーのボリュームを右スピーカーより大きくすることです。そして右にパンさせるということはその逆です。ですので、目いっぱい左にパンさせた時は、左スピーカーからしか音が出ず、右スピーカーから音が出なくなります。その逆も同様です。

したがってパンさせると音が左から右、右から左に移動したように聞えるのである意味3D効果とも言えますが、Direct3Dにはち

ゃんとした3D再生用のAPIが別に実装されているので、もちろん3D効果を出したいのであればそちらのほうを使用します。

パンは、IDirectSoundBuffer:: SetPanメソッドにより簡単に実現できます。サンプルでは次の関数を作成しています。

▼実行結果



```
パン位置は現在0です
```

```
VOID  PanSound(INT  iPan)
{
    g_pDSBuffer->SetPan(iPan);
}
```

簡単ですね。グローバルに用意した、

```
INT  g_iPan=0;
```

をユーザーのキー入力により増減させ、

```
case  VK_LEFT:
    g_iPan-=(g_iPan>DSBPAN_LEFT)＊100;
    break;
case  VK_RIGHT:
    g_iPan+=(g_iPan<DSBPAN_RIGHT)＊100;
    break;
```

毎フレーム、PanSound関数をコールしています。

```
PanSound(g_iPan);
```

左右のパン値の限界値が定義されているので（DSBPAN_LEFT、DSBPAN_RIGHT）それを超えない程度に増減させています。また、範囲が[-10000,10000]なので+-1ずつでは遅すぎて効果を確認し辛く、効果を速く体感できるように100をかけています。

340 周波数を変更する

●IDirectSoundBuffer:: SetFrequencyメソッドにより周波数を変更させる

初級

サンプルプロジェクト　340　周波数変更

周波数の変更はIDirectSoundBuffer::SetFrequencyメソッドにより簡単に実現できます。しかし、何のために周波数を変更するのでしょう？

まずは周波数を変更することによって、どうような効果が生まれるのかを先に理解しましょう。周波数を変更すると“音の高さ”と“音の速さ”の2つの変化が起こります。これをピッチが変化するとも言います。したがって周波数を変更するというこはピッチの変更とも言えます。

音の高さとは、「女の人や子供の声は、成人男性よりも高い」という表現における高さのことです。

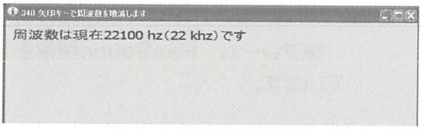
音の速さとは、例えば周波数22khzのサウンドがあり、そのサウンドを全て再生するには1分かかるとしましょう。このサウンドの周波数を倍の44.1khzにするとします。するとサウンドは30秒で再生が終了します。つまりスピードが2倍になったのです。

聡明な読者は気付くかもしれませんが、音の高低と速さの2つの変化は見方を変えた同じ現象とも言えます。その2つの変化は原理的には分離できないものですが、優秀なサウンドソフトでは、速度を変えないで音の高低だけを変化させる（ように見せる）ものがあります。

さて、どのような場合に周波数を変更するのかというと、例えばエンジン音です。たった一つのWAVファイルから周波数を変更することにより、それこそ様々な回転数のエンジン音を簡単に作りだすことができます。本サンプルではその効果を意図して作成しています。周波数が低いときは大型トラックあるいは戦車のディーゼルエンジン、周波数が高いときは低排気量のバイクのような音になります。試してみてください。

サンプルでは次の関数を作成しています。

▼実行結果



周波数は現在22100 hz(22 khz)です

```
VOID  ChangePitch(DWORD  dwFrequency)
{
     g_pDSBuffer->SetFrequency(dwFrequency);
}
```

簡単ですね。グローバルに用意した、

```
DWORD  g_dwFrequency=22100;
```

をユーザーのキー入力により増減させ、

```
case  VK_LEFT:
    g_dwFrequency-=(g_dwFrequency>100)*100;
    break;
case  VK_RIGHT:
    g_dwFrequency+=100;
    break;
```

毎フレーム、ChangePitch関数をコールしています。

```
ChangePitch(g_dwFrequency);
```

341 ボリュームを変更する

●IDirectSoundBuffer:: SetVolumeメソッドによりボリュームを変化させる 初級

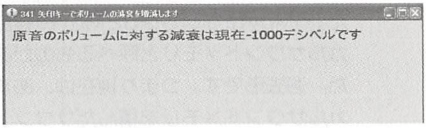
サンプルプロジェクト 341 ボリューム変更

本Tipsのタイトルはボリュームの変更ですが、正確に言うとボリュームに対する減衰の増減です。あくまで原音のボリュームを減衰（小さく）させるかどうかという話であり、原音以上のボリュームを出すものではありません。原音以上のボリュームを出すことは、アンプリテュード（Amplitude）とかアンプリファイ（Amplify）と呼ばれる別の効果でありエフェクトの部類に入ります。

減衰は音圧の単位として一般的なdb（デシベル）で表します(音自体もdbで表しますが)。原音のボリュームをフルに出したい場合は減衰=0dbとします。ボリュームを小さくするということは言い換えると減衰を大きくすることであり、減衰=1000dbなどとすれば1000dbだけ音が小さくなります。

サンプルでは次の関数を作成しています。

▼実行結果


341 矢印キーでボリュームの減衰を増減します

原音のボリュームに対する減衰は現在-1000デシベルです

```
VOID  ChangeVolume(INT  iVolume)
{
    g_pDSBuffer->SetVolume(iVolume);
}
```

簡単ですね。グローバルに用意した、

```
INT  g_iVolume=-1000;
```

をユーザーのキー入力により増減させ、

```
case  VK_LEFT:
      g_iVolume-=(g_iVolume>DSBVOLUME_MIN)*100;
      break;
case  VK_RIGHT:
      g_iVolume+=(g_iVolume<DSBVOLUME_MAX)*100;
      break;
```

毎フレーム、ChangeVolume関数をコールしています。

```
ChangeVolume(g_iVolume);
```

342 サウンドカードのローカルメモリってあるの？

●サウンドカードの変遷とカード上のメモリについて

初級

ビデオカードはカード自体に専用のメモリを積んでいますよね。ローカルビデオメモリと呼ばれるもので、そのメモリ上にリソースを置けばシステムメモリよりも、はるかに高速に処理できるのはご存知でしょう。

ではサウンドカードの場合はどうでしょう？

サウンドカードにもカード自体に搭載された16Mbyteないしは32Mbyte程度のローカルサウンドメモリと呼べるものがありました。過去形です。つまり現在は、あまりローカルサウンドメモリを積んだサウンドカードは存在しません（一部高級クラスに例外はあります）。

昔、サウンドカードの接続方式がISAバスだった時代、ISAバス経由のシステムメモリアクセスが低速なためローカルサウンドメモリのメリットが表面化しましたが、現在はPCI接続によりシステムメモリを使用してもパフォーマンスに影響しないため、ローカルメモリを持つ必要性が無くなったのです。

DirectX　Capsビューアー等でサウンドカードの性能を列挙した際、昔のカードほどローカルビデオメモリの容量が大きい現象を読者の中でも経験した人はいるのではないでしょうか？

現在のサウンドカードでは、ローカルメモリが“ゼロ”と報告されるものがほとんどでしょう。ただ、PCI+システムメモリでも十分とはいえ、特殊なメモリを積むことにより一定のパフォーマンスアップが期待できるのは事実で、玄人指向のカードには、ローカルメ

モリを64Mbyte持っている機種もあったりします。

343 バックグラウンドでも鳴るようにする

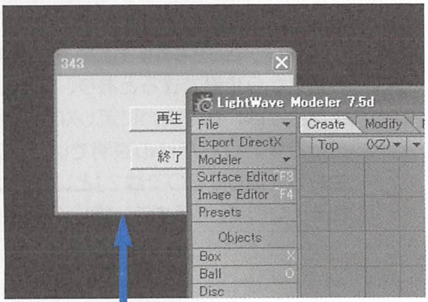
●DSBCAPS_GLOBALFOCUSフラグを指定する　初級

サンプルプロジェクト　343 バックグラウンド再生

特に指定しない場合、DirectSoundアプリは、バックグラウンドになると（フォーカスが他のアプリに移ると）、音が聞えなくなります。

バックグラウンドでも音が聞えるようにする設定は簡単です。セカンダリバッファ作成時に渡すDSBUFFERDESC構造体のdwFlagsメンバにDSBCAPS_GLOBALFOCUSを指定（あるいは追加）すればいいだけです。

▼実行結果


343
再生
終了
LightWave Modeler 7.5d
File
Export DirectX
Modeler
Surface Editor F3
Image Editor F4
Presets
Objects
Box X
Ball O
Disc
Create
Modify
Top
(XZ)

バックグラウンドになると音が聞こえなくなる

```
dsbd.dwFlags = DSBCAPS_GLOBALFOCUS;
```

本サンプルのコード上では次のようにしています。

```
DSBUFFERDESC dsbd;
ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
dsbd.dwSize = sizeof(DSBUFFERDESC);
dsbd.dwFlags = DSBCAPS_CTRLPAN | DSBCAPS_CTRLVOLUME |
DSBCAPS_CTRLFREQUENCY
      | DSBCAPS_GLOBALFOCUS;//このフラグを指定するとバックグラウンドでも音が聞える
dsbd.dwBufferBytes = dwWavSize;
dsbd.guid3DAlgorithm = DS3DALG_DEFAULT;
dsbd.lpwfxFormat = pwfex;

g_pDSound->CreateSoundBuffer( &dsbd, &g_pDSBuffer, NULL );
```

344 ストリーミング再生とは？

●音を出力に流しながら再生すること

初級

インターネットウェブページの動画配信などで“ストリーミング再生”というものがあるのはほとんどの人がご存知でしょう。ストリーミング再生では、データを一括してダウンロードするのではなく、動画を見ている途中に動的に読み込んでいく形になります。DirectSoundのストリーミング再生も仕組みはそれと同じものです。

ストリームStream「流れる、流す」という言葉から想像できるとおり、ストリーミング再生とは、音を出力に流しながら再生することです。これだけの説明では、「普通の再生も音を流してるのでは？」という声が聞えてきそうなので、もうすこし詳しく説明します。

ストリーミング再生において、“何が何処を流れる”のかということを知ると両者の違いが明らかになります。非ストリーミング再生（つまり静的バッファ再生）は、最初にバッファに読み込んだサウンドは最後までバッファ内で変化しません。これに対しストリーミング再生では、バッファの中をサウンドが流れます。言い換えると、バッファの内容は常に変化します。音を再生している裏側で、次の音を読み込み、その音を再生、その再生中にさらに次の音を読み込み…という流れを繰り返します。このようにするとバッファを非常に小さくできます。そしてそれが唯一のメリットです。

345 なぜストリーミングするの？

●巨大なWAVファイルでも再生できる

初級

最近のPC環境では、数十メガバイトもあるWAVファイルを静的バッファ（非ストリーム）で再生できるようです。筆者のPCでは、100メガバイトのWAVファイルを静的バッファで再生してしまったので驚きました。“驚いた”というのは性能に驚いたのではなく、「これではストリーミング再生の意義を書きづらい」と思って驚いた（というか焦った）のです。ところが、160メガバイトのWAVは再生に失敗してくれたので、変な言い方ですが一安心しました。

静的バッファ再生には限界があります。WAVデータのサイズとそっくり同じ分だけのバッファが必要になり、おのずと超巨大なWAVは再生に失敗します。そのような場合には、ストリーミング再生するしか方法はありません。ストリーミング再生では、WAVファイルのサイズに依存しないので、どんなに巨大なWAVファイルでも再生できます。

346 ストリーミング再生はあくまでアプリケーション実装

●ストリーミング再生は直接的には実装されていない

初級

DirectSoundには、ストリーミング再生は直接的には実装されていません。ですので、アプリケーション側で実装しなくてはなりません。

これを残念なことと思う必要はありません。むしろ、ストリーミング再生の普遍的原理を理解、経験する良い機会です。APIが全部やってくれる場合はそれはそれで便利ではありますが、その代償として原理を見失うということはDirectSoundに限ったことではありません。

ストリーミング再生の原理、実装を把握したとしたら、それは非常に価値あることであり、DirectSound以外の分野でも必ず役に立つでしょう。

347 ストリーミングの仕組み

●DirectSoundにおける具体的な実装手順を把握する

上級

実際に実装する際に必要な知識として、具体的な手順を示します。

ストリーミング再生は、大まかに言えば次の手順で行います。

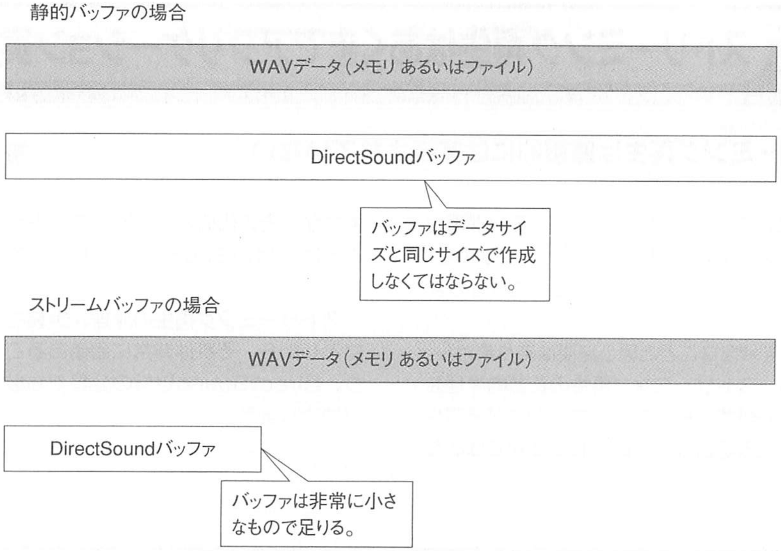
①DirectSoundの初期化
②DirectSoundバッファの作成。バッファはごく小さなサイズで構わない。
　最初にそのバッファ分だけはWAVデータを読み込んでおく。
③再生
④再生中、バッファ内で再生が終わった領域から順次、次のWAVデータを上書きしていく。
③と④を繰り返す。

③と④を繰り返す際に、上手い具合に繋がるようにするのがコツというか、そうしなければなりません。一見すると非常に難しそうですが、これには上手い方法があります。

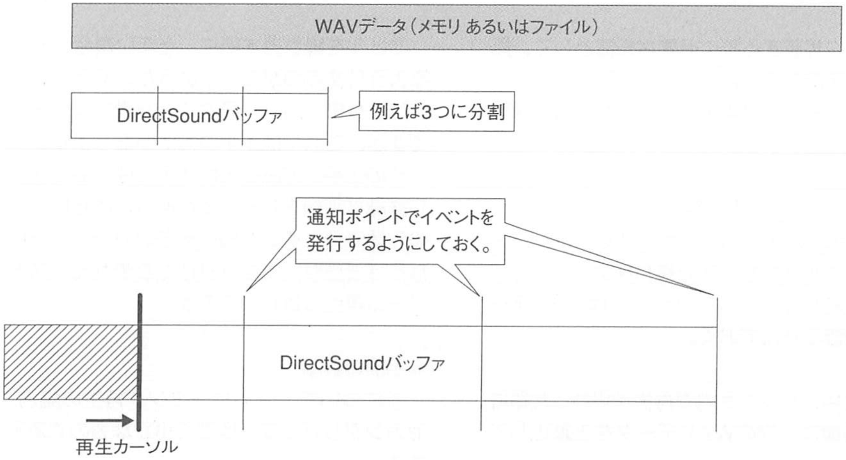
その上手い方法を理解するには、もうすこし詳細な手順を知る必要があり、さらに、その手順で発生する問題点を予想できるようになるほど理解できなければなりません。ストリーム再生は難しいのです。

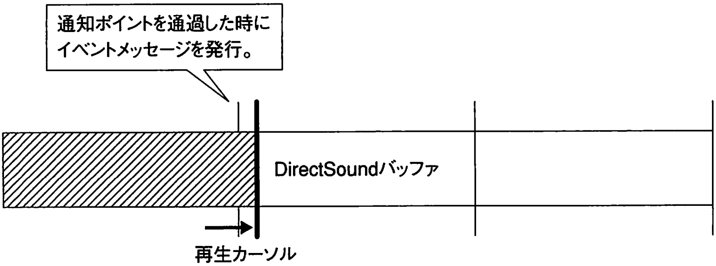
●手順の詳細

②について。ストリーミング再生の場合、セカンダリバッファはごく小さなもので済みます。


静的バッファの場合
WAVデータ（メモリ あるいはファイル）
DirectSoundバッファ
バッファはデータサイズと同じサイズで作成しなくてはならない。
ストリームバッファの場合
WAVデータ（メモリ あるいはファイル）
DirectSoundバッファ
バッファは非常に小さなもので足りる。

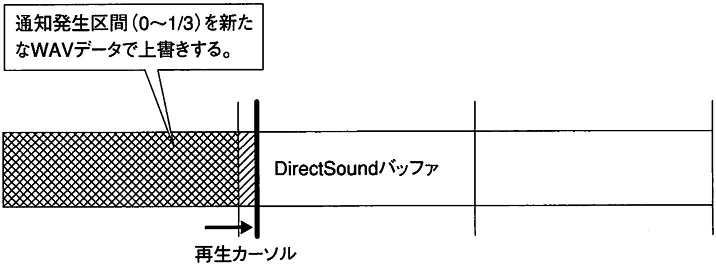
③から④の手順をさらに詳細に示すと次のようになります。


WAVデータ（メモリ あるいはファイル）
DirectSoundバッファ
例えば3つに分割
通知ポイントでイベントを発行するようにしておく。
DirectSoundバッファ
再生カーソル



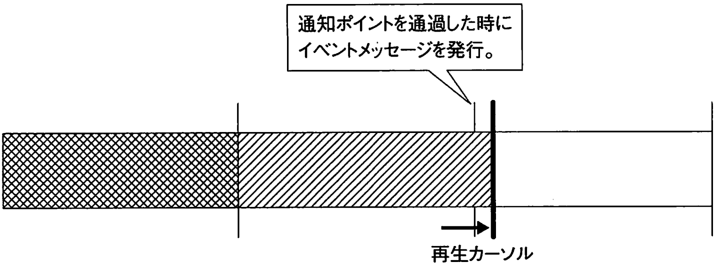
通知ポイントを通過した時にイベントメッセージを発行。

DirectSoundバッファ
再生カーソル

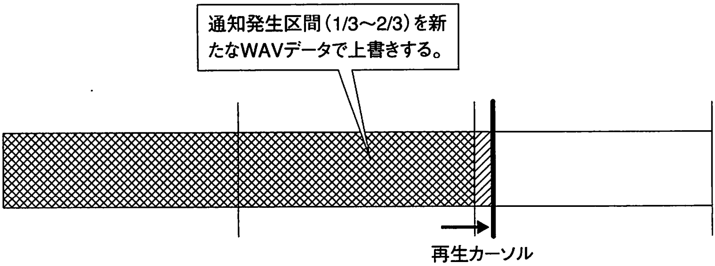


通知発生区間(0〜1/3)を新たなWAVデータで上書きする。

DirectSoundバッファ
再生カーソル

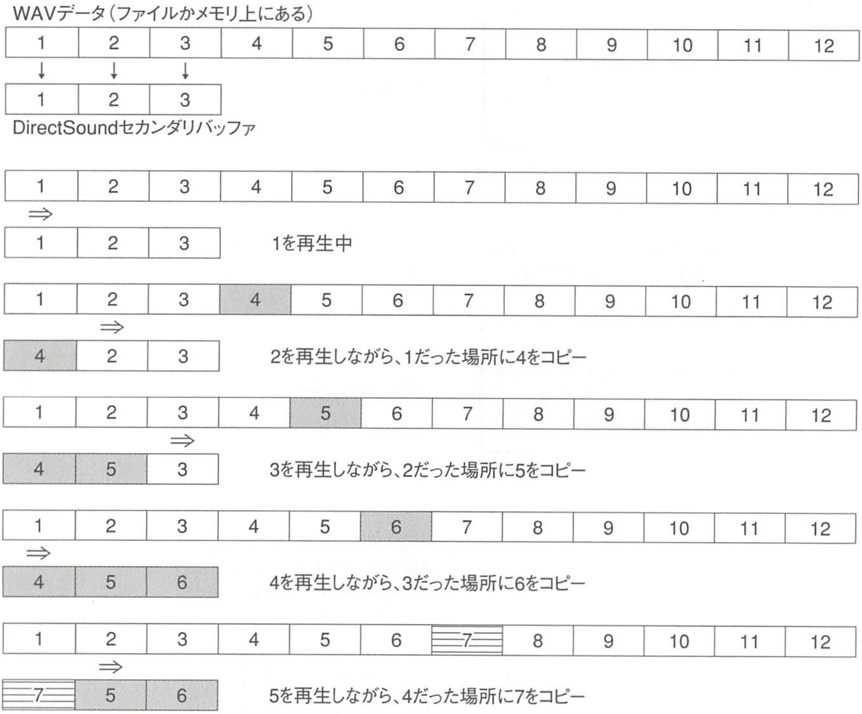


通知ポイントを通過した時にイベントメッセージを発行。



通知発生区間(1/3〜2/3)を新たなWAVデータで上書きする。

この全体的な流れは次のとおりです。


WAVデータ（ファイルかメモリ上にある）
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3
DirectSoundセカンダリバッファ
1を再生中
4 2 3
2を再生しながら、1だった場所に4をコピー
4 5 3
3を再生しながら、2だった場所に5をコピー
4 5 6
4を再生しながら、3だった場所に6をコピー
7 5 6
5を再生しながら、4だった場所に7をコピー

One Point

本書のストリーミング再生サンプルにおいては、バッファサイズがロックするサイズのぴったり整数倍（これがミソ）になっていて、（そのようにバッファを作成していて）、終端をまたいだロックが起こり得ません。このことは非常に便利で、書き込みカーソルが不要になります。

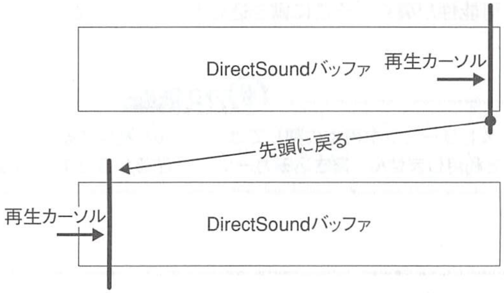
348 ラップアラウンドとは？

●ここでは、再生カーソルがバッファの最終ポイントを越えた時、先頭に戻ること

初級

ラップアラウンド（wrap-around）とは、「回り込み」「取り囲む」とか「カーブした」という意味で、コンピューターにおいては、メモリーラップアラウンド（メモリーが上限までいくとアドレス0に戻ること）、ラップアラウンド・リスト（循環リスト）などと使用される用語です。

ここで言うラップアラウンドもこれと同じことであり、再生カーソルがバッファの最終ポイントを越えた時、バッファの先頭にまた戻ることを意味します。したがって、ラップアラウンドしながら再生カーソルはバッファ内をぐるぐると何周もすることになります。


DirectSoundバッファ
再生カーソル
先頭に戻る
再生カーソル
DirectSoundバッファ

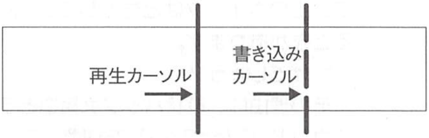
349 再生カーソルと書き込みカーソルって何？

●アドレスを指す動くポインター

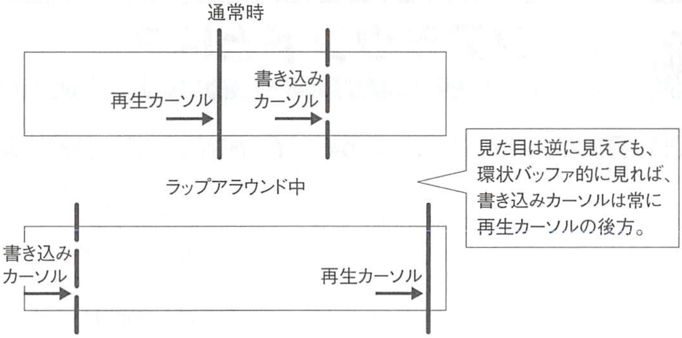
中級

そもそもカーソルとは「走るもの」という意味で、メモリ上では「アドレスを指す動くポインター」という意味になります。

再生カーソルは、現在再生しているWAVデータの位置（アドレス）を表します。書き込みカーソルは、WAVデータを書き込んでもいい安全な領域を示す地点（アドレス）を示します。


再生カーソル
書き込み
カーソル

書き込みカーソルは、再生カーソルのやや後方（時間的に後）にあります。ラップアラウンド中は、この位置関係が見た目で逆になりますが、論理的な位置関係は同じです。


通常時
再生カーソル
書き込み
カーソル
見た目は逆に見えても、
環状バッファ的に見れば、
書き込みカーソルは常に
再生カーソルの後方。
ラップアラウンド中
書き込み
カーソル
再生カーソル

再生カーソルと書き込みカーソルの間にデータを書き込んではいけません。その区間にあるWAVデータは既にDirectSoundがロードしている可能性が高く、そこに書き込むと大抵の場合は音飛び等の不具合が発生します。書き込む場所は、書き込みカーソルより後方だけです。そのための書き込みカーソルなのですから。

One Point

ただ、本書でのストリーミング再生に関しては、書き込みカーソルを利用しません。書き込みカーソルを利用するよりもシンプル、かつ、上手い方法があるからです（Tips347、355、356を参照）。

350 なぜ2つのポインターが帰ってくるの？

●環状バッファのロックについて理解する

中級

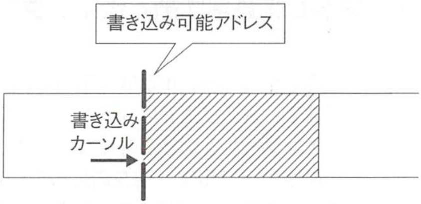
バッファにWAVデータを書き込み時は、IDirectSoundBuffer::Lockメソッドを実行するわけですが、このメソッドが返すバッファ上のポインタはときとして2つ帰ってくるときがあります。

なぜでしょう？

その理由は、環状バッファ構造とラップアラウンド（Tips348）を理解していれば容易に想像がつきます。

まず一つだけ帰ってくる場合は次の場合です。

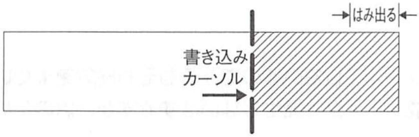
▼バッファ終端まで十分に余裕がある


書き込み可能アドレス
書き込み
カーソル

しかし、もし書き込みカーソルがバッファの終端付近だとどうなるでしょう？　書き込みカーソルから終端までのサイズよりも、ロックしたいサイズのほうが大きい場合です。

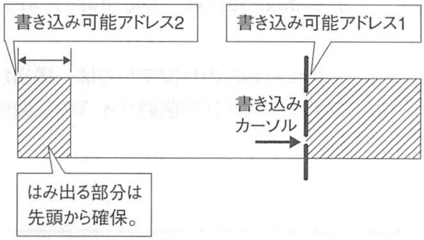
このような場合は、ロックされる領域は2つの離れた領域になります（それらの領域の合計）。

▼書き込みカーソルがバッファ終端付近にある



したがって2つの領域それぞれのポインターを知る必要があり、これがポインターが2つ帰ってくる理由です。

▼ロックされるのは2つの離れた領域



One Point

本書のストリーミング再生サンプルにおいては、返されるポインターは常に一つです。なぜなら、バッファサイズがロックするサイズのぴったり整数倍（これがミソ）になっていて、（そのようにバッファを作成していて）、終端をまたいだロックが起こり得ないからです。また、書き込みカーソルも使用していません。

351 環状バッファのコツ

ポインタに剰余演算を行っておく

中級

環状バッファ（ring buffer リングバッファ）は、バッファの終端までポインタが進んだ時、次のポインタをバッファの先頭に戻すラップアラウンド（Tips348）を行うことにより、小さな容量で大きなデータを扱えるようにするバッファです。

ストリーム再生を行う際のDirectSoundバッファは、もちろん環状バッファとして扱います。環状バッファで、終端を越えるような領域を確保しようとすると、領域を2分割して確保するしかありません（Tips350）。そのような状況にならないようにするための小技を紹介します。

何らかのバッファのポインタを進めた場合は、そのポインタに常に剰余演算（%演算子による）を行っておきます。これをすることにより、バッファの終端を越えないことが保障されます。

```
Pointer+=n;
Pointer  %=  BufferSize;
```

サンプルでも次のようにしています。

```
g_pDSBuffer->Unlock(pv,dwSize,NULL,NULL);
g_dwNextOffset+=dwSize;
g_dwNextOffset  %=g_dwBufferSize;
```

もっともこのサンプルでは、最初からバッファサイズを通知区間サイズの整数倍で作成しているので、そもそも終端をまたいだロックは発生しないはずですが、念のためです。

One Point

環状バッファは、循環バッファとも呼ばれます。

352 書き込みカーソルは混乱のもと！

書き込みカーソルの必要性を考える

中級

ストリーミング再生するために、書き込みカーソルは必ずしも必要ではありません。実際、本書サンプルは書き込みカーソルを利用していません。

書き込みカーソルがどれくらいのオフセットなのかが不定のため、逆に書き込みカーソルによりロックしてしまうと、いちいちポインタを補正しなくてはならなくなり、大変面倒です。せっかくバッファサイズと通知区間サイズを合わせているのに、整数倍ではない地点からロックしてしまうことになり、その結果ロック領域が2つになってしまいます。

確かに、書き込みカーソルが有用な場合もあるでしょうが、少なくとも本書において書き込みカーソルを考えてしまうことは、混乱のもとです。

353 WAVデータ上の“読み込みカーソル”が必要！

読み込みカーソルという概念を理解する

中級

「読み込みカーソル」という用語はヘルプドキュメントには登場しませんが、重要な概念です。これが無くてはストリーミング再生が出来ません。なぜなら、WAVファイル上

のデータを「今どの部分まで読み込んでいるのか」ということを記録しておかないと、つまり現在まで読み込んだWAVファイル上のポインタを持っていないと、ストリーミング再生は不可能と言えます。

既に説明した格好になりますが、読み込みカーソルとは、現在までのWAVデータを読み込んだ地点です。言い換えると、次に読み込むべきWAVデータ上の地点のことです。

サンプルでは読み込みカーソルは、

```
DWORD  g_dwCurRead
```

として定義しています。

354 メモリー括方式とファイル逐次方式

本書のストリーミングサンプル2つのタイプを理解する

上級

本書において、ストリーミング再生サンプルは2種類作成しました。

1つは、WAVデータを一旦全てユーザーメモリ（アプリ側で確保するシステムメモリ）に読み込んでしまいます。ストリーミング中は、そのシステムメモリからWAVデータをこまめに取り出します。つまり初期化を終えた後は、ファイルに用はありません。WAVデータはすべてシステムメモリから取り出します。

2つめは、WAVデータを最初はバッファサイズ分しか読み込まず、ストリーミング中にWAVファイルからWAVデータをこまめに読み込むものです。ストリーミング中にシステムメモリバッファは一切使用しません。すべてファイルから取り出します。

両者はお互いに一長一短があります。

1の方法は、パフォーマンスの低下がありませんが、WAVデータ分のユーザーメモリを用意する必要があります。また、システムメモリに全WAVデータを読み込むので、WAVデータが巨大な場合は、初期化時にそれなりに時間がかかります。

2の方法は、システムメモリを食わないということと、初期化時に読み込むのはセカンダリバッファ分だけなので一定時間しか掛からないというメリットがあります。

しかし、一定間隔でファイルアクセスが発生するので、動的に読み込むサイズが大きければ、パフォーマンスが低下する可能性をはらんでいます（ファイルアクセスはメモリアクセスに比べ超低速ですから）。

どちらの手法を採用するかは個々の状況を鑑みて判断することになるでしょう。

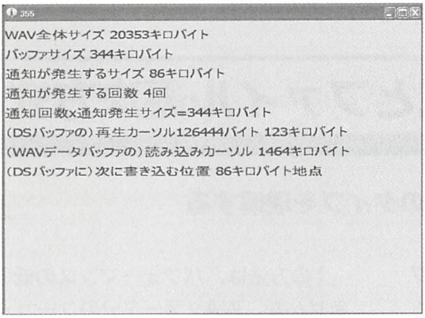
355 ストリーミング再生を行う（メモリー括保存バージョン）

●WAVデータを初期化時に全てメモリに読み込む方式

上級

サンプルプロジェクト　355 ストリーミング（メモリ）

▼実行結果



```
WAV全体サイズ 20353キロバイト
バッファサイズ 344キロバイト
通知が発生するサイズ 86キロバイト
通知が発生する回数 4回
通知回数x通知発生サイズ=344キロバイト
(DSバッファの)再生カーソル126444バイト 123キロバイト
(WAVデータバッファの)読み込みカーソル 1464キロバイト
(DSバッファに)次に書き込む位置 86キロバイト地点
```

本サンプルではDirect3Dを使用していますが、パラメーターの情報表示用であって、もちろん本質ではありません。別ファイルにまとめてあるので、Direct3Dは意識しないようにしてください。

さて、本サンプルはコード量が多いので基本的には「コードを見てください」的なノリにならざるを得ませんが、本サンプルではほとんどの変数、特に主要な変数は全てグローバルに宣言しているので、それらの意味を示しておけば、コードを理解しやすくなるものと思います。

なお、大まかな流れはTips347で示していますので、実装時の具体的な流れについて触れたいと思います。

●具体的な流れ

キーポイントとなるのは2つの関数、CreateStreamBuffer関数とWriteStream関数だけです。

CreateStreamBuffer関数は初期化時のキーポイント、WriteStream関数はストリーミング再生中のキーポイントです。

【初期化時】

ウィンドウ初期化

↓

Direct3D初期化

↓

DirectSound初期化

↓

通知用イベントと通知用スレッドの作成

↓

WAVデータ読み込み（CreateStreamBuffer関数）
WAVファイルを開き、データを読み込む。

↓

DirectSoundバッファの作成
まずは通知区間のサイズから先に算出する。サイズの決定はファイルから取得したフォーマットを基に行う。決め打ちで決定はしない。DirectSoundバッファを通知区間の倍数サイズで作成。（サンプルのデフォルトでは4倍）

↓

DirectSoundバッファにフォーマット及びWAVデータをコピーする。コピーするWAVデータは先頭からDirectSoundバッファサイズ　分。

↓

DirectSound通知インターフェイス（IDirectSoundNotify）を取得する。

↓

通知区間の数だけ要素を持つ配列を用意し、

それぞれの要素を通知区間のサイズで初期化。その配列をDirectSound通知インターフェイスのメソッド（SetNotificationPositions）に渡して、通知区間情報を1DirectSoundに登録する。

【ストリーミング再生時】
再生カーソルが通知区間を通過した瞬間にNotifyProc関数にメッセージが来る。

↓

WriteStream関数に次の書き込みオフセットを渡して実行。書き込みオフセットとはDirectSoundバッファ上に書き込むアドレス。

↓

通知区間のサイズ分だけ、DirectSoundバッファにWAVデータをコピー。次の書き込みオフセットを通知区間のサイズ分だけ進める。

大まかに言えばこんなところです。

●グローバル変数の意味

LPDIRECTSOUNDBUFFER g_pDSBuffer=NULL;
DirectSoundセカンダリーバッファオブジェクトのポインター。

HANDLE g_hNotifyEvent=NULL;
通知イベントのハンドル。

HANDLE g_hNotifyThread=NULL;
通知スレッドのハンドル。

BYTE＊ g_pbWavData=0;
WAVデータを保存しているユーザーメモリ（システムメモリ）領域のポインタ。

DWORD g_dwDataSize=0;
WAVデータのサイズ（バイト単位）。したがって、ユーザーメモリのサイズでもあります。

DWORD g_dwBufferSize=0;
DirectSoundバッファのサイズ（バイト単位）。

DWORD g_dwCurPlay=0;
DirectSoundバッファ内の再生カーソル

DWORD g_dwCurRead=0;
ユーザーメモリにあるWAVデータからの読み込みカーソル（Tips353）。

DWORD g_dwNumNotify=4;
イベントメッセージの通知回数。表現を変えるとDirectSoundバッファの分割数。

DWORD g_dwNotifySize=0;
イベントメッセージを発生する間隔のサイズ（バイト単位）。言い換えると、DirectSoundバッファ内の各分割区域ひとつ分のサイズ。

DWORD g_dwNextOffset=0;
DirectSoundバッファ内、次に書き込むオフセット。初期化が終わった直後はDirectSoundバッファのサイズと一致。その後は通知区間サイズ分だけ増加してゆく。

BOOL g_boEndOfData=FALSE;
WAVデータを最後まで読み込んだときにTRUEになる。このようなフラグを用意しておかないと、サウンドの最後付近がおかしなことになる。

356 完全なストリーミング再生を行う（ファイル逐次読み込みバージョン）

WAVデータをファイルからその都度読み込む方式

上級

サンプルプロジェクト　356 ストリーミング（ファイル）

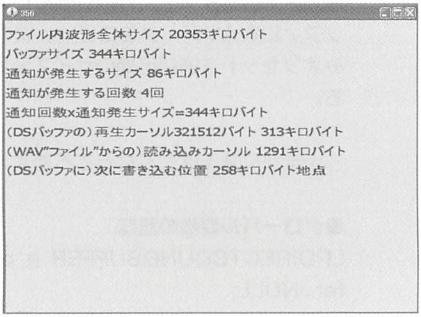
サウンドが数100メガあるような場合で、ユーザーメモリーすら容量不足になる環境では、このサンプルのほうが有用だと思います。ましてやギガ単位である場合、選択肢はこのサンプルしかないことになります。

原理はTips355と全く同じです。異なるのはWAVデータを取り出すのがユーザーメモリではなくファイルになったという点だけです。コード上の相違点もそれに伴った部分であり、若干異なる程度です。

BYTE＊　g_pbWavData=0;
が指すメモリ領域には、メモリ版では全てのWAVデータを格納しましたが、通知区間分のWAVデータのみを格納します。WAV全体よりも小さいのはもちろん、DirectSoundバッファよりもさらに小さな領域になります。

次のコードは99%メモリ版と同じですが、2行目のmemcpyの第2引数が異なります。メモリ版では、システムメモリのポインタを渡していただけですが、ここではReadWavFile関数になっています。

▼実行結果



```
ファイル内波形全体サイズ 20353キロバイト
バッファサイズ 344キロバイト
通知が発生するサイズ 86キロバイト
通知が発生する回数 4回
通知回数x通知発生サイズ=344キロバイト
(DSバッファの)再生カーソル321512バイト 313キロバイト
(WAV"ファイル"からの)読み込みカーソル 1291キロバイト
(DSバッファに)次に書き込む位置 258キロバイト地点
```

```
hr=g_pDSBuffer->Lock(dwOffset,g_dwNotifySize,&pv,&dwSize,NULL,NULL,0);

memcpy(pv,ReadWavFile(g_dwCurRead,szFileName),dwSize);
g_dwNextOffset+=dwSize;
g_dwNextOffset  %=g_dwBufferSize;
//読み込みカーソルを進める
g_dwCurRead+=dwSize;

hr=g_pDSBuffer->Unlock(pv,dwSize,NULL,NULL);
```

ReadWavFile関数は、読み込みカーソルとファイル名を指定すれば、そのファイルのそのオフセットからWAVデータを一時的なメモリ上に読み込み、そのポインタを返すようにしました。

実装は次のとおりです。

```
BYTE＊  ReadWavFile(DWORD  dwOffset,LPSTR  szFileName)
{
    FILE＊  fp=fopen(szFileName,"rb");

    fseek(fp,g_dwDataOffset+dwOffset,SEEK_SET);

    fread(g_pbWavData,1,g_dwNotifySize,fp);//まずは、波形データを読み込み

    fclose(fp);

    return  g_pbWavData;
}
```

エフェクト全般

357 エフェクトとは？

原音に何らかの2次的加工を施すこと

初級

エフェクトとは、原音（エフェクト前のサウンド）に何らかの2次的加工を施すことです。

エコーをかければサウンドが幾重にも“こだま”のように再生されるようになり、コーラスをかければサウンドが透き通ったように変化し、また、ディストーションをかければ、バリ…バリ…と音割れのようにヘビーメタル調になったりします。

このように積極的に原音を変化させるタイプの他に、意図した帯域をカットまたは増幅させることにより微調整的な加工をするものもエフェクトと呼ばれます（フィルター系エフェクト）。

DirectSoundのエフェクトは、現実のギターのエフェクターと似ています。現実のエフェクターは基本的に一つのエフェクトに対し一個のエフェクターがあり、一個につき7、8千円から高いもので1万数千円したりしますが（20年前から価格が変わらない！）、それらの主要なエフェクターが全て実装されているのに“ただ”で使用できるわけですから、お得なものと言えるでしょう（楽器店に行ってエフェクターを見てみると面白いですよ）。

さて、エフェクトは可能な限りハードウェア（サウンドカード）で実現されますが、ハードウェアが当該エフェクトをサポートしていない場合はソフトウェア的にエミュレートされます。オンボードのサウンドハードウェアは一般的に低機能なので、特にリバーブ系の高度なエフェクトはサポートしていないようです。

何れにしても、プログラムから見ればハードウェア性能がどうであれ、エフェクトをかけることはできます。

358 エフェクトを分類してみる

●エフェクトのタイプを把握する

初級

●遅延・反響系

【エコー：Echo】

これについては説明するまでもないと思いますが念のため説明しておきます。エコーとは「反響」の意味です。原音より遅延させた音をミックスすることにより、サウンドが“こだま”のように何重にも重なって聞こえるようになります。

【コーラス：Chorus】

元のサウンドと若干遅らせたエコーサウンドをさらに変調して音程をずらします。それを元のサウンドと重ね合わせることにより和音のような効果が生まれ“クリアな音”、“透明感のある音”になります。

歌番組でよく耳にすると思いますが、ボーカルの生声よりも明らかに透き通った音声に加工されていると感じる場合は、コーラスエフェクトが効いていると思っていいでしょう。

【フランジ：Flange】

コーラスエフェクトと原理は同じです。ただし、サウンドがエコーバックされるまでの遅延時間をコーラスよりもさらに短くすることにより元のサウンドとの干渉を生み出し、それにより独特な効果がかかります。調整次第でロボットの声みたいな効果を作ることもできます。

【I3DL2リバーブ】

IA-SIG（Interactive Audio Special Interest Group）というサウンド業界団体が発表したI3DL2（Interactive 3-D Audio Level2）という仕様に準拠したリバーブです。

【Wavesリバーブ】

環境リバーブにはI3DL2 リバーブともう1つ、このWavesリバーブがあります。

ハード・ソフトシンセサイザーの雄、米Waves社からライセンス取得したMaxxVerb技術をDMO（DirectX以外のコンポーネントだが、DirectXから利用できるCOMオブジェクト）として組み込んだもので、DirectSoundからも利用できます。

●歪み（ひずみ）系

【ディストーション Distortion】

ヘビーメタルサウンドでも多用されるエフェクトで、いわゆる「音が割れる」という現象を意図的に作り出すものです。割れた音とは、スピーカーのピークを超えた入力をしたときの「バリバリ」音のことです。

もともと“音割れ”はオーディオにとって排除すべきものなのですが、1970年代に「かっこいい音」として受け入れられたことにより、エフェクトとなりました。個人的には「反社会的な音」「不良っぽい音」になるような気がします。

●うねり系

【ガーグル　Gargle】

ガーグルを日本語で言うと「うがい」という意味です。そのまま商品名になっている“うがい薬”をご存知の人も多いでしょう。文字通り、“うがい“をしながら話しているかのようなエフェクトがかかります。コーラスやフランジもLFOにより”うねり“を加えますが、遅延・反響という効果の印象が強いことから、そちらの分類に入れています。

●フィルター系

【コンプレッション Compression】

このエフェクトの原理としては、指定した入力（音圧db）を超えた入力部分を圧縮により減衰させることです。用途としては、音のアタックを強調したいときや、音割れを防ぐとき、および、各楽器（時にはボーカル）のボリューム差を少なくして各楽器が均等に聞こえるようにするときなどに使用します。

【パラメトリック イコライザ Parametric Equalizer】

イコライザーの意味は、「録音時の音と再生時の音をイコール（同じ）にする」ことです。もともとは、音を録音した現場と、音を再生する場所で同じように聞こえる音を出力するためのものなので、積極的に音を変えるというエフェクトというよりも補正的な意味合いが強いものです。

パラメトリック・イコライザーは略して「パライコ」と呼ばれますが、イコライザーはこの他に、「グライコ」グラフィック・イコライザーというものもあります。一般的に思いつくのはグライコのほうではないでしょうか。グライコは知ってのとおり、固定された周波数帯域ごとに上下にスライドするスライダー（フェーダー）が5,6個、多いもので30個ほど付いているものです。一方パライコは、ダイアルが3,4個しかないものが一般的です。

パライコもグライコも倍音構成を変えるという機能は、ほぼ同じと考えて差し支えありません。両者の違いは、パライコは周波数帯域が固定ではなく自由に移動できるという点で、より細かな変更が可能であるのに対し、グライコは周波数帯域が固定されている（決められている）ので、より“補正”という意味合いが強いものの、ツマミ（フェーダー）の位置関係から倍音構成を視覚的（グラフィック）に読み取りやすいという特徴があります。

One Point

現実のエフェクターでは、フランジはフランジャー、コンプレッションはコンプレッサーと呼ばれ、微妙に呼び名が異なります。

359 ディレイ系とは？

●サウンドを“遅延“させる

初級

サウンドを“遅延“させることを基本原理にしているエフェクトです。

エコー、コーラス、フランジがこれにあたり、さらに、より高度なディレイ系エフェクトにリバーブがあります。

リバーブは、原理としてはエコーエフェクト等と同じですが、それらが単なる反響であるのに対しリバーブは“現実の音の反射のシミュレート”を目的としているもので、例えば“あたかもコンサートホールで聞いているような”感覚にさせる“残響（リバーブ）”を生み出すものです。原理を直感的に表現すると無数のディレイ音の集合であり、ある意味エコーエフェクトの最終進化形態といえるでしょうか。また実現がもっとも難しいエフェクトであると言われています。

360 歪み（ひずみ）系とは？

●“音割れ”に似た効果を作り出す

初級

音圧レベルを増幅することにより意図的にピークを引き上げ、“音割れ”に似た効果を作り出します。

ディストーションがこれにあたります。

361 うねり系とは？

●音を“揺らす””うねらせる“効果を生み出す

初級

時間周期的な変化を与えて、音を“揺らす”、”うねらせる“効果を生み出します。

ガーグルがこれにあたります。

362 フィルター系とは？

●意図した帯域の成分を選択して通す、あるいは通さないこと

初級

フィルター系を他のエフェクトのように「加工」的な意味合いで捉えると本質を見失います。　エフェクターは何らかの加工を施すものなので、すべてがフィルター系になってしまいます。このフィルターの意味は「ろ過」的な意味で考えるべきです。つまり意図した帯域の成分を選択して通す、あるいは通さないことです。

コンプレッション、パラメトリック・イコライザーがこれにあたります。

363 セカンダリバッファにエフェクトをかける

●IDirectSoundBuffer::SetFXメソッドによりエフェクトを適用する 初級

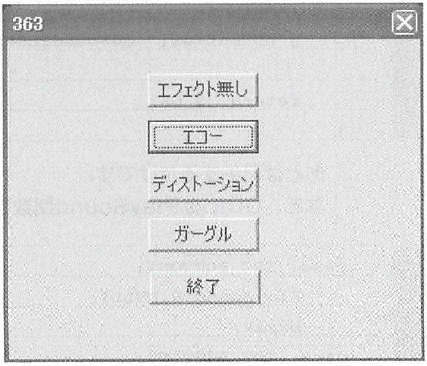
サンプルプロジェクト 363 エフェクト再生

●最初に

本サンプルを含め、本Tips以降のほとんどのDirectSoundサンプルはDirectSoundバッファを配列で用意します。

実際に読み込むWAVファイルは一つなので配列にする必要が無い場合がほとんどですが、他のプロジェクトへの流用を考慮して、より再利用しやすい形で設計したためです。

▼実行結果


363

```
LPDIRECTSOUNDBUFFER8  g_pDSBuffer[  MAX_SOUND+1  ];
```

●コード解説

エフェクトをかけているコード上の場所はPlaySound関数内です。

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID*  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();
```

エフェクトをかける瞬間は、バッファを停止しなければエフェクトの適用が失敗するので、停止します。

```
    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  *pGuid;
    g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult);
```

まず、DSEFFECTDESC構造体を意図したエフェクトとして初期化することからはじめます。

dsEffect.guidDSFXClassにはかけたいエフェクトの種類に応じてGUID識別子を指定します。このサンプルでは3つの異なるエフェクトを選択でき、それぞれのエフェクトはそれぞれのGUIDにより識別されます。

引数として受け取ったpGuidに3つのエフェクトどれかのGUIDが入っています。

DSEFFECTDESC構造体のメンバを埋めたら、バッファのメソッドSetFXを実行します。これによりエフェクトが適用されます。

第1引数はエフェクトの数、第2引数はエフェクト構造体のアドレスです。

```
        //再生する
        g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);

        return  S_OK;
}
```

あとは再生するだけです。

なお、GUIDはPlaySound関数コール時に、それぞれのGUIDを渡しています。

```
    case  IDC_BUTTON1:
        PlaySound(0,NULL);
        break;
    case  IDC_BUTTON2:
        PlaySound(0,&GUID_DSFX_STANDARD_ECHO);
    return  TRUE;
    case  IDC_BUTTON3:
        PlaySound(0,&GUID_DSFX_STANDARD_DISTORTION);
    return  TRUE;
    case  IDC_BUTTON4:
        PlaySound(0,&GUID_DSFX_STANDARD_GARGLE);
    return  TRUE;
```

364 セカンダリーバッファのエフェクトを解除する

●IDirectSoundBuffer::SetFXメソッドによりエフェクトを解除する　初級

参考サンプルプロジェクト　363　エフェクト再生

DirectSoundバッファのエフェクトを全て取り去るには、IDirectSoundBuffer::SetFXの第1引数にゼロを渡してコールします。

363サンプルでは次のようにしています。

```
g_pDSBuffer[  dwSoundIndex  ]->SetFX(0,NULL,NULL);
```

エフェクトを取り去るだけなので、原音に戻るだけで、サウンド自体はそのまま鳴り続けます。

365 複数のエフェクトを重ねる

●エフェクト構造体を配列で用意・初期化し、SetFXを実行する

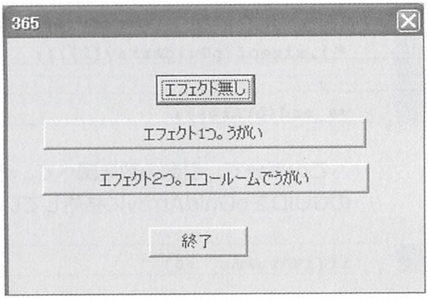
中級

サンプルプロジェクト　365　多重エフェクト

●最初に

本サンプルを含め、本書のほとんどのDirectSoundサンプルはDirectSoundバッファを配列で用意します。実際に読み込むWAVファイルは一つなので配列にする必要が無い場合がほとんどですが、より再利用しやすい形で設計したためです。

▼実行結果



```
LPDIRECTSOUNDBUFFER8  g_pDSBuffer[  MAX_SOUND+1  ];
```

●コード解説

エフェクトは単一でかけるだけではなく何種類かのエフェクトを同時に組み合わせてかけることもできます。例えばディストーションをかけた音にフランジャーをかけて全く別の音にするなどという動的な音質変換が可能になります。

エフェクト再生およびエフェクトを重ねる処理はPlaySound関数で全て行っています。

```
HRESULT  PlaySound(DWORD  dwSoundIndex,  ...)
```

PlaySound関数の仕組みは簡単で、可変引数により渡されたエフェクトを全て重ねるというものです。呼び出し側は好きなだけエフェクト識別子であるGUIDを渡すことがで

きますので、この関数は他のプロジェクトでも重宝するでしょう。

エフェクト引数の数を格納するための変数です（すべての引数の数ではありません）。

```
INT  iVArgAmt=0;
```

エフェクト識別用のGUIDを最大8個格納するための配列です。

```
const  GUID＊  pGuidArray[8];
```

```
va_list  pVArgs;
va_start(pVArgs,  dwSoundIndex);
  while(    va_arg(  pVArgs,  const  GUID＊)  !=  NULL  )
  {
    iVArgAmt++;
  }
  va_start(pVArgs,  dwSoundIndex);
  for(DWORD  i=0;i<iVArgAmt;i++)
  {
                    memcpy(&pGuidArray[i],&va_arg(pVArgs,const              UID
＊),sizeof(pGuidArray[i]));
  }
va_end(pVArgs);
```

この部分で、引数の数を調べて、それぞれのGUIDをpGuidArrayに格納しています。

```
if(iVArgAmt  >0)
```

iVArgAmtは全引数の中にエフェクトGUID引数が何個あるかを表しますので、その数は全引数-2となります。-2とする理由は、第1引数dwSoundIndexはカウントしませんし、最後にターミネーターとして機能させているNULLもカウントしていないからです。したがって、エフェクトGUIDが引数に無い場合はiVArgAmtがゼロになります。

```
DSEFFECTDESC＊  dsEffectArray  =  new  DSEFFECTDESC[iVArgAmt];
```

DirectSoundでは、多重エフェクトを設定する場合はDSEFFECTDESC構造体を配列に、正確に言えば連続して格納したメモリの先頭アドレスを渡すようになっています。iVArgAmt の数だけ（エフェクトの数だけ）DSEFFECTDESC 型のメモリを確保して、その先頭アドレスをdsEffectArrayに代入しています。

その下のforループブロックで、各要素を設定しています。設定といっても、エフェクトごとにGUIDを格納しているだけです。

DSEFFECTDESC　型の配列を用意して

しまえば、あとはごく簡単です。Tips363と同じようにIDirectSoundBuffer::SetFXメソッドをコールするだけです。

```
g_pDSBuffer[  dwSoundIndex  ]->SetFX(iVArgAmt,dsEffectArray,dwResult);
```

第1引数はエフェクトの数、第2引数は全エフェクト構造体配列の先頭アドレスです。

コーラス

366 コーラスとは？

●LFOにより周期的に変化させ、音程をも変化させる

初級

音を遅らせるという点でエコーと原理は同じです。コーラスの場合は、その遅らせる時間をLFOにより周期的に変化させて音程をも変化させるという点においてエコーより複雑です。

オシレーターとは発信器のことであり、そもそもシンセサイザーが音を生成する仕組みは発信器によるものです。LFO（Low Frequency Oscillator）ロー・フリークエンシー・オシレーターは低周波発信器のことであり、ここでの使われ方のように、原音を“うねらせる・ゆらす”際に使用されます。

エコーは原音とディレイ音及びそのミックス音しかありませんでしたが、このコーラスの場合は、それらに加えてLFOも考えてパラメーターを決定しなくてはなりません。いくつかの異なる音程となった各音を原音にフィードバックさせて混ぜ合わせるので現実のコーラスと同様に和音効果で透明感のあるクリアーな音になります。

367 コーラスをかける

●サウンドにコーラスエフェクトをかける

初級

サンプルプロジェクト　367 コーラス

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_CHORUSを指定して、SetFXメソッドを実行します。

●コーラスエフェクトのセット

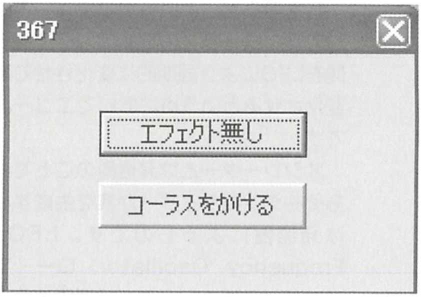
```
case IDC_BUTTON2:
    PlaySound(0,&GUID_DSFX_STANDARD_CHORUS);
```

```
HRESULT PlaySound(DWORD dwSoundIndex,const GUID* pGuid)
{
    g_pDSBuffer[ dwSoundIndex ]->Stop();

    DSEFFECTDESC dsEffect;
    DWORD dwResult=NULL;
    memset(&dsEffect, 0, sizeof(DSEFFECTDESC));
    dsEffect.dwSize = sizeof(DSEFFECTDESC);
    dsEffect.dwFlags = 0;
    dsEffect.guidDSFXClass = *pGuid;
    if(FAILED(g_pDSBuffer[ dwSoundIndex ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[ dwSoundIndex ]->Play(NULL,0,NULL);
```

これでとりあえずコーラスエフェクトがかかります。

何も指定しないとデフォルトのコーラスがかかりますが、好きなようにコーラスの特徴を指定することができます。それについては次のTipsを参考にしてください。

▼実行結果



368 コーラス効果を調整する

IDirectSoundFXChorus8:: SetAllParametersメソッドでコーラス特性を設定する　初級

サンプルプロジェクト　367　コーラス

コーラス特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXChorus8インターフェイスを取得することから始めます。IDirectSoundFXChorus8を取得したら、DSFXChorus構造体を用意し、メンバを好きなように設定

します。

DSFXChorus構造体を初期化し終えたらIDirectSoundFXChorus8::SetAllParametersメソッドで設定し、完了です。

●コーラス特有のパラメーター設定

```
IDirectSoundFXChorus8＊  pChorus;
pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_CHORUS,
          0,IID_IDirectSoundFXChorus8,(VOID＊＊)&pChorus);

    DSFXChorus  dsChorus;
    //  ドライ～ウェット（比率）  0～100  0は完全なドライつまり全く処理しない（コーラスをかけない）ことになる
    dsChorus.fWetDryMix=100;
    //LFO(Low  Frequency  Oscillator) により”うねり”を加える比率   0～100
    dsChorus.fDepth=15;
    //フィードバック（比率）-99～99
    dsChorus.fFeedback=45;
    //LFOの周波数  0～10
    dsChorus.fFrequency=1.3f;
    //LFOのうねり波形の形状。DSFXCHORUS_WAVE_TRIANGLE（三角波）かDSFXCHORUS_WAVE_SIN（サイン波）のいづれか。
    dsChorus.lWaveform=DSFXCHORUS_WAVE_SIN;
    //  位相差をラジアン角で指定。取り得る値は  1/2ラジアンごとに0,1,2,3,4の５種類のみ。
    dsChorus.lPhase=DSFXCHORUS_PHASE_90;
    //原音から作成したディレイ音のディレイタイム  0～20
    dsChorus.fDelay=20;

    //エフェクトパラメーターをセット
    pChorus->SetAllParameters(&dsChorus);

    pChorus->Release();
```

設定が終わったら、もはやIDirectSoundFXChorus8に用はないのでリリースしましょう。

●パラメータ用の構造体

```
typedef  struct  _DSFXChorus  {
    FLOAT       fWetDryMix;
    FLOAT       fDepth;
    FLOAT       fFeedback;
    FLOAT       fFrequency;
    LONG         lWaveform;
    FLOAT       fDelay;
    LONG        lPhase;
}  DSFXChorus,  *LPDSFXChorus;
```

●メンバの意味

fWetDryMix

　ドライとウェットの比率を100分率で指定します。ドライとはコーラスがかからない原音を意味し、ウェットはコーラス音を意味します。0で完全なドライ（全くコーラスをかけない原音そのまま）、100で完全なウェット（コーラス音）となります。

fDepth

　LFOにより音をどれだけ“うねらせるか”を指定します。この値を大きくすれば、音の揺れ具合が大きくなりますが、揺らしすぎると別のエフェクトになってしまいます。

fFeedback

　コーラスは、原音とコーラス音を混ぜ合わせて出力しますが、これはその比率を−99〜99の間で指定します。フィードバックが高いほど「ぼやけた」音になります。

fFrequency

　LFOの周波数、つまり、1秒間に何個の波を原音にぶつけるかということです。デフォルトでは1.1（1.1Hz）になっていることからもLFOが（かなり）“低周波”であることが分かるでしょう。LFOは音を発生させるというよりも“なんらかの周期”を与えるためのものです。この値を大きくすると“うねり”が大きくなります。

lWaveform

　LFOの波の形状です。サイン波か三角波のどちらか1つを指定します。

fDelay

　コーラス音をどれだけ遅らせるかをミリ秒単位で指定します。エコーよりも速いディレイタイムにするのが一般的です。

lPhase

　Phaseフェースとは「位相」のことです。位相とは波形の時間的な位置のことです。位相のみでエフェクトをかけるフェイザーというエフェクターもあります。位相は通常、角度で位置を表します。このパラメーターは90度ごとに5段階の値をとりますが、ラジアン角で指定するので、0から4までの数値の何れかを指定します。

コンプレッション

369 コンプレッションとは？

●音を圧縮するもの

初級

　コンプレッサーというエフェクターは、ダイナミックレンジ（音量）を操作することからダイナミクス系という分類もされます。コンプレッサーという名前から、このエフェク

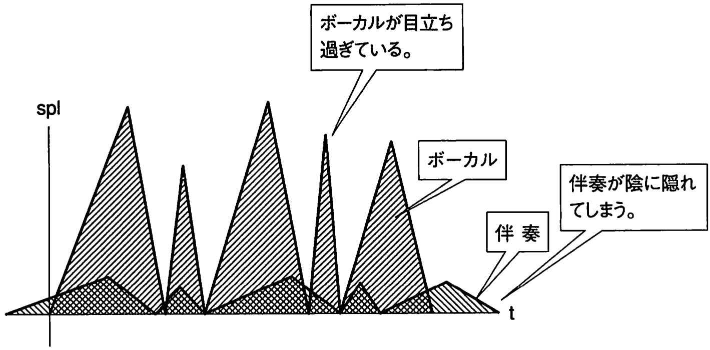
トが音を圧縮するものだということは想像できると思います。

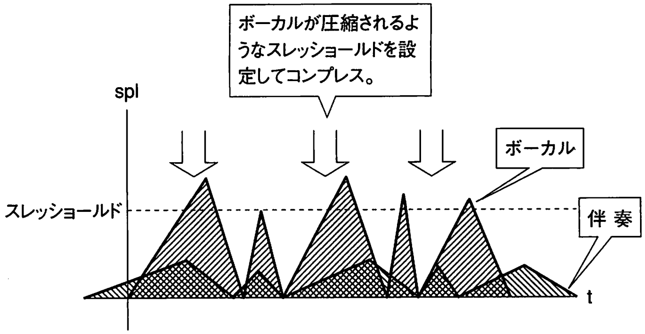
しかし、何のために音を圧縮するのでしょう？　圧縮したからといってどうなるのでしょう？

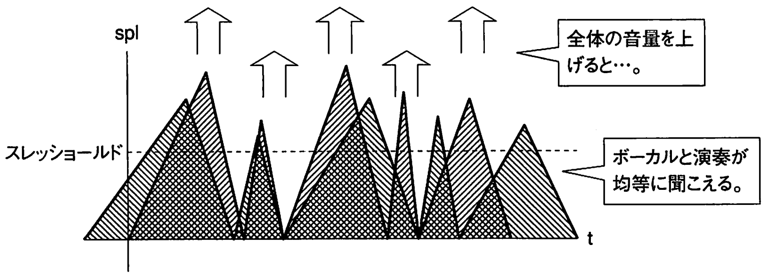
例えば、ボーカルと伴奏の波形が図のようになっているとします。

この図から分かることは、ボーカルは音の起伏が激しく、伴奏はある程度一定の音量であるということです。そして問題なのは、全体を通して聞くとボーカルの陰に伴奏が隠れてしまっているように聞えることです。要はボーカルが目立ち過ぎるのです。こんな時にコンプレッサーが役に立ちます。

コンプレッサーは、指定した“しきい値“(threshold　スレッショールド）より大きい音量部分を圧縮します。圧縮された部分の音量は下がります。


ボーカルが目立ち過ぎている。
spl
ボーカル
伴奏が陰に隠れてしまう。
伴 奏
t


ボーカルが圧縮されるようなスレッショールドを設定してコンプレス。
spl
ボーカル
スレッショールド
伴 奏
t


spl
全体の音量を上げると…。
スレッショールド
ボーカルと演奏が均等に聞こえる。
t

370 コンプレッションをかける

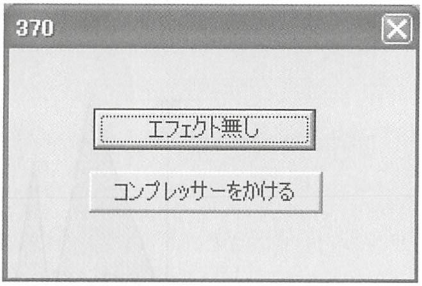
●サウンドにコンプレッションエフェクトをかける

初級

サンプルプロジェクト　370 コンプレッション

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_COMPRESSORを指定して、SetFXメソッドを実行します。

▼実行結果



●コンプレッションエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_COMPRESSOR);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID*  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  *pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずコンプレッションエフェクトがかかります。

何も指定しないとデフォルトのコンプレッションがかかりますが、好きなようにコンプレッションの特徴を指定することができます。次のTipsを参考にしてください。

371 コンプレッション効果を調整する

●IDirectSoundFXCompressor8:: SetAllParametersメソッドでコンプレッション特性を設定する 初級

サンプルプロジェクト 370 コンプレッション

コンプレッション特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXCompressor8インターフェイスを取得することから始めます。

IDirectSoundFXCompressor8を取得したら、DSFXCompressor構造体を用意し、メンバを好きなように設定します。

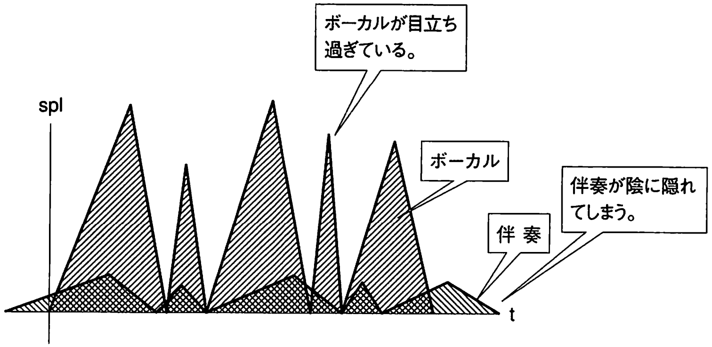
DSFXCompressor構造体を初期化し終えたらIDirectSoundFXCompressor8::SetAllParametersメソッドで設定し、完了です。

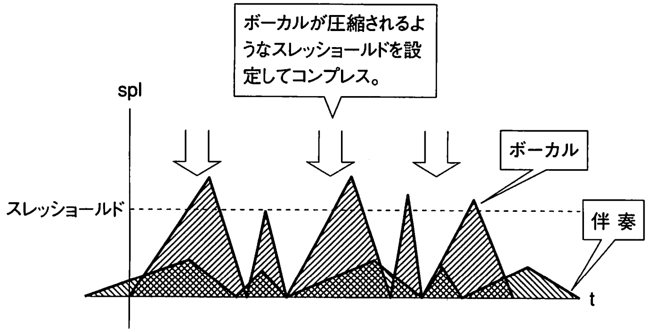
●コンプレッション特有のパラメーター設定

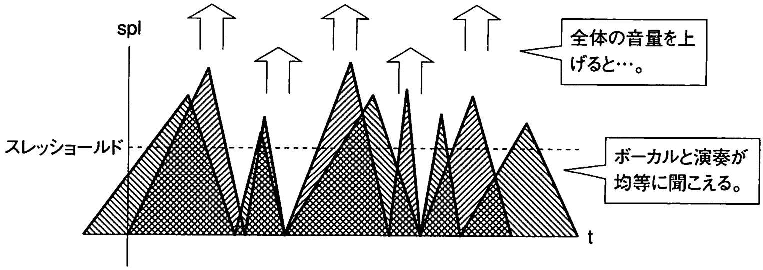
```
IDirectSoundFXCompressor8＊  pCompressor;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_COMPRESSOR,
    0,IID_IDirectSoundFXCompressor8,(VOID＊＊)&pCompressor)))
{
    return  E_FAIL;
}
DSFXCompressor  dsCompressor;
//-60～60 （dB）
dsCompressor.fGain=25;
//0.01～500(ミリ秒)
dsCompressor.fAttack=0.1;
//50～3000 （ミリ秒）
dsCompressor.fRelease=150;
//-60～0 （dB）
dsCompressor.fThreshold=-45;
//1～100 （n  :  m）
dsCompressor.fRatio=100;
//0～4 （ミリ秒）
dsCompressor.fPredelay=0;
//エフェクトパラメーターをセット
if(FAILED(pCompressor->SetAllParameters(&dsCompressor)))
{
    pCompressor->Release();
    return  E_FAIL;
}
pCompressor->Release();
```

設定が終わったら、もはやIDirectSoundFXCompressor8に用はないのでリリースしましょう。

パラメーターの意味を見る前に、事前知識が必要です。もう一度“ボーカルが大きすぎるサウンド”の図を見てください。


ボーカルが目立ち過ぎている。
spl
ボーカル
伴奏が陰に隠れてしまう。
伴 奏
t

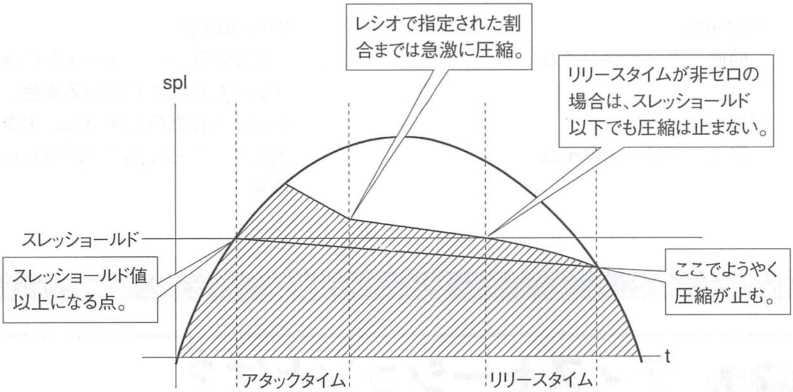

ボーカルが圧縮されるようなスレッショールドを設定してコンプレス。
spl
ボーカル
スレッショールド
伴 奏
t


spl
全体の音量を上げると…。
スレッショールド
ボーカルと演奏が均等に聞こえる。
t

仕組みは、伴奏に対して相対的に大きな音量であるボーカルの飛び出ている音量部分を圧縮により“つぶし”ます。そうすることによって、ボーカルと伴奏の音量差が小さくなります。それによって相対的に伴奏を表に引っ張り出すことを意味します。コンプレスしたあとに全体のボリュームを上げれば、ボーカルの音量が過度に上がることなく伴奏を聴きやすくすることができるわけです。これがコンプレッサーの目的の一つです。

また、カラオケなどでは（特に酒の入った）素人がマイクに大声を張り上げて歌うことはよくあります。そういった場合に全体の音量を均等に“均す（ならす）”のも実はコンプレッサーです。

コンプレッサーのパラメーターは、図で見ると次のようになっています。なお、この図はアタックタイム、リリースタイムともに非ゼロで、レシオ（圧縮比率）を3:1にしている場合の図例です。


レシオで指定された割合までは急激に圧縮。
spl
リリースタイムが非ゼロの場合は、スレッショールド以下でも圧縮は止まない。
スレッショールド
スレッショールド値以上になる点。
ここでようやく圧縮が止む。
t
アタックタイム
リリースタイム

スレッショールド：Threshold

圧縮する音量レベルの“しきい値”です。スレッショールド以上の音量が圧縮対象となります。

アタックタイム：Attack Time

音量がスレッショールドを越えた瞬間から実際に圧縮がレシオで設定されたレベルまで圧縮されるまでの時間。

リリースタイム：Release Time

スレッショールド以下になってから圧縮が止むまでの時間。

ゲイン：Gain

圧縮された音量。

レシオ：Ratio

圧縮比率。ある瞬間において、スレッショールドから圧縮前の音量最高点までを…a、スレッショールドから圧縮後の音量最高点を…bとした時にa:bである比率。

●パラメータ用の構造体

```
typedef  struct  _DSFXCompressor  {
    FLOAT    fGain;
    FLOAT    fAttack;
    FLOAT    fRelease;
    FLOAT    fThreshold;
    FLOAT    fRatio;
    FLOAT    fPredelay;
}  DSFXCompressor,  *LPDSFXCompressor;
```

●メンバの意味

fGain

　圧縮後の音の音量。コンプレスした後に音量を上げたいことはよくある事なので便利なパラメーターです。

fAttack

　前述、アタックタイム

fRelease

　前述、リリースタイム

fThreshold

　前述、スレッショールド

fRatio

　前述、レシオ

fPredelay

　音量がスレッショールドに達してもプリディレイが設定されている場合、すぐにアタックタイムに進行しません。アタックタイムに加えて、さらにもう1段階遅れを用意しています。

372 ディストーションとは？

●“音割れ”に似た効果を出すもの

初級

　ディストーションは、オーディオにとって本来害悪である“音割れ”に似た効果を出すものです。音割れを体感することは簡単です、フリーの波形編集ソフトでマイク入力を目一杯上げて大きめに声を録音すれば、大抵は音割れを起こします。失礼な言い方をすれば、読者のPCスピーカーが安物であればなおさら音割れを発生させることができます。

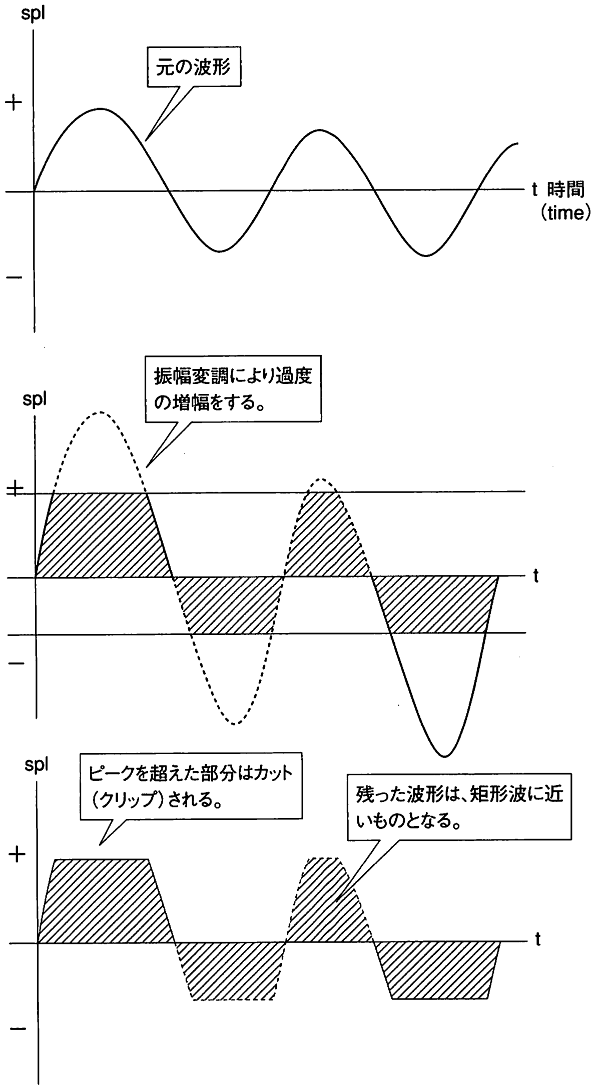
　オーディオの歴史は、この“音割れ”との戦いという側面と、他方でこのディストーションやファズ、オーバードライブ等のエフェクターの世界では“音割れのクオリティ”を追求してきたという側面があるのは面白いことです。

　ディストーションの原理は、音割れの発生メカニズムを知れば、おのずと明らかになります。

　音割れはなぜ発生するのでしょう？

　音の出力デバイス（スピーカー）には音圧レベルの上限があり、それはピークと呼ばれます。入力レベルでピークを超えている部分があっても、出力ではピークまでしか出力されません。入力がピークを超えている時に“割れた音”となって聞えるわけです。

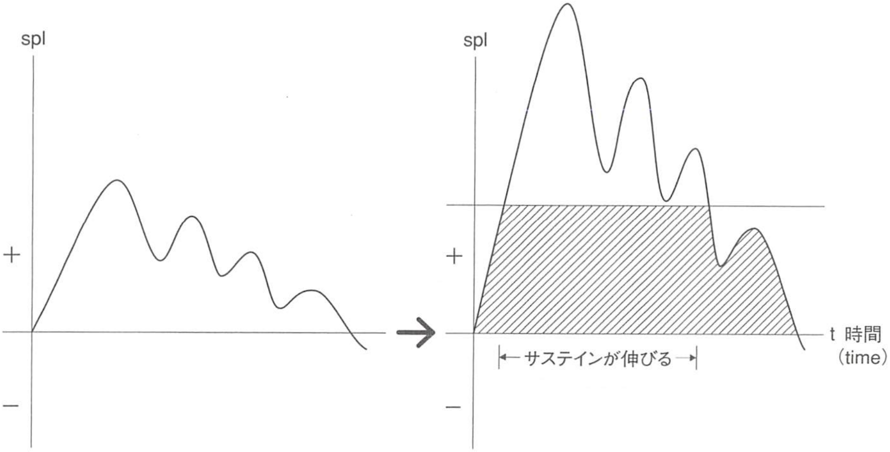
　もうお分かりかと思いますが、ディストーションなどの歪み系エフェクターは入力レベルを故意にピーク以上に上げることによって歪み音を出します。


spl
元の波形
+
t 時間
（time）
−
spl
振幅変調により過度の増幅をする。
+
t
−
spl
ピークを超えた部分はカット（クリップ）される。
残った波形は、矩形波に近いものとなる。
+
t
−

このような処理を施すと、波形は矩形波に近くなります。矩形波のPSG音源が“バリバリ”音である理由もこのためです。変な言い方ですが、PSG音源は素の状態でディストーションエフェクトがかかっているようなものです。

また、音圧レベルを上げることにより、歪むことはもちろん、一般的に歪み系エフェクトをかけるとボリュームも上がったように聞えます。実際、グラフから明らかなように音量の面積、それこそボリューム（領域）は増えています。

また、ギターのような減衰音（音量がだんだん小さくなる）でも、ディストーションをかけるとサステイン（音量が一定な部分）が持続します。これが、ヘビーメタルサウンドでよく聞きますが、ギター音が「ギュイィ～ィィン！」と長く伸びる理由です。


spl
+
−
spl
+
−
t 時間
（time）
サステインが伸びる

373 ディストーションをかける

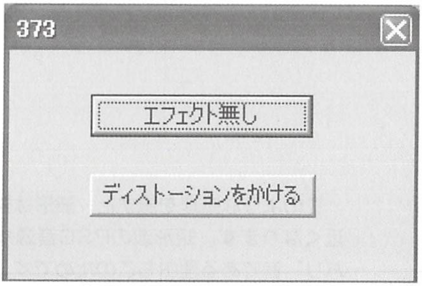
●サウンドにディストーションエフェクトをかける

初級

サンプルプロジェクト　373 ディストーション

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_DISTORTIONを指定して、SetFXメソッドを実行します。

▼実行結果


373
エフェクト無し
ディストーションをかける

●ディストーションエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_DISTORTION);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID＊  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  ＊pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずディストーションエフェクトがかかります。

何も指定しないとデフォルトのディストーションがかかりますが、好きなようにディストーションの特徴を指定することができます。それについては、次のTipsを参考にしてください。

374 ディストーション効果を調整する

●IDirectSoundFXDistortion8:: SetAllParametersメソッドでディストーション特性を設定する　初級

サンプルプロジェクト　373 ディストーション

ディストーション特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXDistortion8インターフェイスを取得することから始めます。

IDirectSoundFXDistortion8を取得したら、DSFXDistortion構造体を用意し、メンバを好きなように設定します。

DSFXDistortion構造体の初期化終了後、IDirectSoundFXDistortion8:: SetAllParametersメソッドで設定し、完了です。

●ディストーション特有のパラメーター設定

```
DirectSoundFXDistortion8＊  pDistortion;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_DISTORTION,
    0,IID_IDirectSoundFXDistortion8,(VOID＊＊)&pDistortion)))
{
    return  E_FAIL;
}
```

```
DSFXDistortion  dsDistortion;
//-60～0（dB）
dsDistortion.fGain=-30;
//０～100（%）
dsDistortion.fEdge=25;
//100～8000（Hz）
dsDistortion.fPostEQCenterFrequency=8000;
//100～8000（Hz）
dsDistortion.fPostEQBandwidth=2000;
//100～8000（Hz）
dsDistortion.fPreLowpassCutoff=8000;
//エフェクトパラメーターをセット
if(FAILED(pDistortion->SetAllParameters(&dsDistortion)))
{
    pDistortion->Release();
    return  E_FAIL;
}
pDistortion->Release();
```

設定が終わったら、もはやIDirectSoundFXDistortion8に用はないのでリリースしましょう。

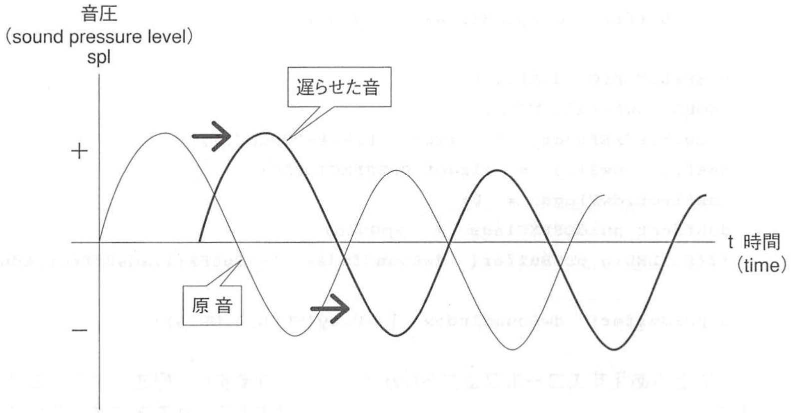
エコー

375 エコーとは？

●もっとも単純なディレイ系エフェクト

初級

DirectXのエコーは、様態としてはステレオフィードバックディレイです。遅延・反響系のエフェクターの基本となる“ディレイ”というエフェクターが存在しますが、コーラスやリバーブ、フランジャー（フランジ）はディレイ系と言えます。そして、エコーはこの中ではもっとも単純なディレイ系エフェクトでしょう。


音圧
(sound pressure level)
spl
遅らせた音
+
t 時間
(time)
原音
−

376 エコーをかける

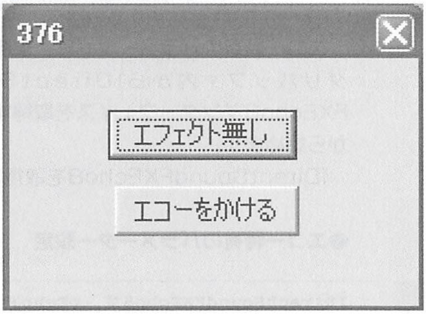
●サウンドにエコーエフェクトをかける

初級

サンプルプロジェクト 376 エコー

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_ECHOを指定して、SetFXメソッドを実行します。

▼実行結果


376
エフェクト無し
エコーをかける

●エコーエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_ECHO);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID*  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  ＊pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずエコーエフェクトがかかります。

何も指定しないとデフォルトのエコーがかかりますが、好きなようにエコーの特徴を指定することができます。それについては、次のTipsを参考にしてください。

377 エコー効果を調整する

●IDirectSoundFXEcho8:: SetAllParametersメソッドでエコー特性を設定する　初級

サンプルプロジェクト　376　エコー

エコー特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXEcho8インターフェイスを取得することから始めます。

IDirectSoundFXEcho8を取得したら、DSFXEcho構造体を用意し、メンバを好きなように設定します。DSFXEcho構造体を初期化し終えたらIDirectSoundFXEcho8:: SetAllParametersメソッドで設定し、完了です。

●エコー特有のパラメーター設定

```
IDirectSoundFXEcho8＊  pEcho;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_ECHO,
    0,IID_IDirectSoundFXEcho8,(VOID＊＊)&pEcho)))
{
    return  E_FAIL;
}
DSFXEcho  dsEcho;
//  ドライ～ウェット（比率）  0～100  0は完全なドライつまり全く処理しない（エコーをかけない）ことになる
```

```
dsEcho.fWetDryMix=100;
//  フィードバック（比率）0～100  0は原音にエコー音を混ぜないことを意味する、100にすると聞えるのは”エコー音”のみ
dsEcho.fFeedback=50;
//  左チャンネル  ディレイ（ミリ秒）1～2000  どれだけ音を遅らせるかという意味
dsEcho.fLeftDelay=300;
//右チャンネル  ディレイ（ミリ秒）1～2000  どれだけ音を遅らせるかという意味
dsEcho.fRightDelay=300;
//  パン  ディレイ（左右チャンネルのスワップ：入れ替えの比率）0  or  1  左右のチャンネルの音を入れ替える
dsEcho.lPanDelay=0;
//エフェクトパラメーターをセット
if(FAILED(pEcho->SetAllParameters(&dsEcho)))
{
    pEcho->Release();
    return  E_FAIL;
}
pEcho->Release();
```

設定が終わったら、もはやIDirectSoundFXEcho8に用はないのでリリースしましょう。

●パラメータ用の構造体

エコーのエフェクトパラメーターは、DSFXEcho構造体により渡します。DSFXEcho構造体の定義は次のとおりです。

```
typedef  struct  _DSFXEcho
{
    FLOAT    fWetDryMix;
    FLOAT    fFeedback;
    FLOAT    fLeftDelay;
    FLOAT    fRightDelay;
    LONG      lPanDelay;
}  DSFXEcho,  *LPDSFXEcho;
```

●メンバの意味

fWetDryMix

ドライとウェットの比率を100分率で指定します。ドライとはエコーがかからない原音を意味し、ウェットはエコー音を意味します。0で完全なドライ（全くエコーをかけない原音そのまま）、100で完全なウェット（エコー音）となります。

fFeedback

エコーは、原音とエコー音を混ぜ合わせて出力しますが、これはその比率を100分率で指定します。フィードバックが高いほど「ぼやけた」音になります。

0は全く混ぜないことを意味するので、その場合原音のみの出力となります。逆に100

にして全てをフィードバックすると音を出力しない設計になっているようなので、事実上の最大は100ではなく99です。

fLeftDelay、fRightDelay

どれだけ音を遅らせるか、その遅延時間をミリ秒で指定します。範囲は1ミリ秒から2000ミリ秒までです。

フランジャー

378 フランジャーとは？

●コーラスとの差異は使う側の設定次第

初級

フランジの仕組みは、構造体を見てもらえば明らかなようにコーラスと同じです。

メンバもコーラスと同一です。コーラスとフランジを分けるのは、パラメーターの値、つまり使う側の設定次第なので、パラメーターの設定によっては、コーラスインターフェイスでフランジ効果を生むこともできますし、またその逆も同様です。

実際、同じパラメーター値を渡すと、コーラスもフランジも同じエフェクトになります。フランジはディレイタイムを短く、LFOによる揺さぶりを大きくしたものです。マイクロソフトのヘルプファイルでもフランジのパラメーターデフォルト値は、そのことを物語っています。

379 フランジャーをかける

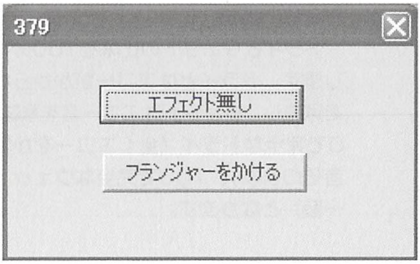
●サウンドにフランジャーエフェクトをかける

初級

サンプルプロジェクト　379 フランジャー

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_FLANGEを指定して、SetFXメソッドを実行します。

▼実行結果



●フランジャーエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_FLANGER);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID＊  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  ＊pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずフランジャーエフェクトがかかります。

何も指定しないとデフォルトのフランジャーがかかりますが、好きなようにフランジャーの特徴を指定することができます。それについては、次のTipsを参考にしてください。

380 フランジャー効果を調整する

IDirectSoundFXFlanger8:: SetAllParametersメソッドでフランジャー特性を設定する 初級

サンプルプロジェクト 379 フランジャー

フランジャー特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXFlanger8インターフェイスを取得することから始めます。

IDirectSoundFXFlanger8を取得したら、DSFXFlanger構造体を用意し、メンバを好きなように設定します。

DSFXFlanger構造体を初期化し終えたらIDirectSoundFXFlanger8::SetAllParametersメソッドで設定し、完了です。

●フランジャー特有のパラメーター設定

```
IDirectSoundFXFlanger8＊  pFlanger;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_FLANGER,
        0,IID_IDirectSoundFXFlanger8,(VOID＊＊)&pFlanger)))
    {
        return  E_FAIL;
    }
    DSFXFlanger  dsFlanger;
    //  ドライ～ウェット（比率） 0～100  0は完全なドライつまり全く処理しない（フランジをかけない）ことになる
    dsFlanger.fWetDryMix=100;
    //LFO(Low  Frequency  Oscillator) により”うねり”を加える比率   0～100
    dsFlanger.fDepth=100;
    //フィードバック（比率）-99～99
    dsFlanger.fFeedback=60;
    //LFOの周波数  0～10
    dsFlanger.fFrequency=10;
    //LFOのうねり波形の形状。DSFXFLANGER_WAVE_TRIANGLE（三角波）かDSFXFLANGER_WAVE_SIN（サイン波）のいづれか。
    dsFlanger.lWaveform=DSFXFLANGER_WAVE_SIN;
    //  位相差をラジアン角で指定。取り得る値は  1/2ラジアンごとに0,1,2,3,4の５種類のみ。
    dsFlanger.lPhase=1;
    //原音から作成したディレイ音のディレイタイム  0～4
    dsFlanger.fDelay=3;
    //エフェクトパラメーターをセット
    if(FAILED(pFlanger->SetAllParameters(&dsFlanger)))
    {
        pFlanger->Release();
        return  E_FAIL;
    }
    pFlanger->Release();
```

設定が終わったら、もはやIDirectSoundFXFlanger8に用はないのでリリースしましょう。

●パラメータ用の構造体

```
typedef  struct  _DSFXFlanger  {
    FLOAT    fWetDryMix;
    FLOAT    fDepth;
    FLOAT    fFeedback;
    FLOAT    fFrequency;
    LONG      lWaveform;
    FLOAT    fDelay;
    LONG      lPhase;
}  DSFXFlanger,  ＊LPDSFXFlanger;
```

●メンバの意味

コーラスと同一です。

ガーグル

381 ガーグルとは？

●“うがい”音のような効果を生む

初級

ガーグルについてヘルプドキュメントには“振幅変調”と書かれていますが、なんのことだか分かりませんね。

ガーグルは、音の波形の振幅すなわち音圧・ボリュームに周期的な変化を加える、つまり、音量に揺らぎ・揺さぶりを加えるエフェクトという意味です。音量が一定の周期で大きくなったり、小さくなったりすることで、“うがい”音のような効果を得るわけです。ガーグルエフェクトは“うがい”の意味ですが、“扇風機の向こう側で喋っているような”エフェクトとも言えます。

382 ガーグルをかける

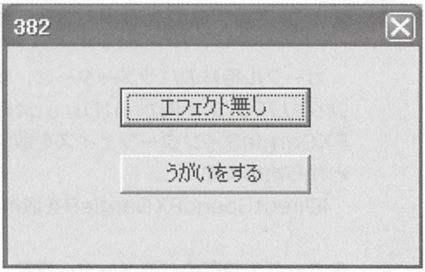
●サウンドにガーグルエフェクトをかける

初級

サンプルプロジェクト　382 ガーグル

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_GARGLEを指定して、SetFXメソッドを実行します。

▼実行結果



●ガーグルエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_GARGLE);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID＊  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  ＊pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずガーグルエフェクトがかかります。

何も指定しないとデフォルトのガーグルがかかりますが、好きなようにガーグルの特徴を指定することができます。それについては、次のTipsを参考にしてください。

383 ガーグル効果を調整する

●IDirectSoundFXGargle8:: SetAllParametersメソッドでガーグル特性を設定する　初級

サンプルプロジェクト　382 ガーグル

ガーグル特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXGargle8インターフェイスを取得することから始めます。

IDirectSoundFXGargle8を取得したら、DSFXGargle構造体を用意し、メンバを好きなように設定します。

DSFXGargle構造体の初期化終了後、IDirectSoundFXGargle8:: SetAllParametersメソッドで設定し、完了です。

●ガーグル特有のパラメーター設定

```
IDirectSoundFXGargle8＊  pGargle;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_GARGLE,
    0,IID_IDirectSoundFXGargle8,(VOID＊＊)&pGargle)))
  {
    return  E_FAIL;
  }
  DSFXGargle  dsGargle;
  //  1～1000 (Hz)
  dsGargle.dwRateHz=17;
  //DSFXGARGLE_WAVE_SQUARE（矩形波）  又は  DSFXGARGLE_WAVE_TRIANGLE（三角波）
  dsGargle.dwWaveShape=DSFXGARGLE_WAVE_TRIANGLE;

  //エフェクトパラメーターをセット
  if(FAILED(pGargle->SetAllParameters(&dsGargle)))
  {
    pGargle->Release();
    return  E_FAIL;
  }
  pGargle->Release();
```

設定が終わったら、もはやIDirectSoundFXGargle8に用はないのでリリースしましょう。

●パラメータ用の構造体

```
  typedef  struct  _DSFXGargle  {
    DWORD    dwRateHz;
    DWORD    dwWaveShape;
  } DSFXGargle,  ＊LPDSFXGargle;
```

●メンバの意味

dwRateHz

音量を上げ下げることを1秒間に何回行うか。すなわち揺さぶりの周波数を指定。500以上あたりから、“揺さぶり“というより音程の変化に聞える。

dwWaveShape

三角波か四角（矩形）波かを指定する。矩形波にすると音がPSG音源のように汚くなる。三角波はより滑らか。

384 ウェーブス・リバーブとは？

●リバーブとは“残響”のこと

初級

リバーブは原音の「残響」を発生させるエフェクトです。残響とは、例えば部屋の中で、なんらかの音を鳴らした際に壁や床及び家具などによって原音が吸収・反射される音のことです。同じ原音でも、環境によって“聞え方”は変わります。

原理を直感的に表現すると無数のディレイ音の集合となります。ただ、目指している音が「現実の残響音」なので、他のエフェクトが単に非現実の面白い音になればそれで良しとするのに対し、残響は現実のものであり人々は現実のものと比べてしまいます。それ故その実現は困難なものであり現在も研究過程にあるエフェクトです。

リバーブは、単なるエフェクトというよりも、もっと高度なシミュレートというべきものなのです。ユーザーが聞いているそれぞれの環境での残響をシミュレートすることは事実上不可能であるので、リバーブの性能はどこまで多くの環境に対応した残響を出せるか、どれだけ広範囲なプリセットを持っているかということになると思います。

Wavesとは、Waves社のライセンスを受けているリバーブだからです。なお、WavesリバーブはDirectSoundの一部ではありません。あくまでDirectSoundからも利用できるものであって、実態はDMOにより提供されるものです。

385 ウェーブス・リバーブをかける

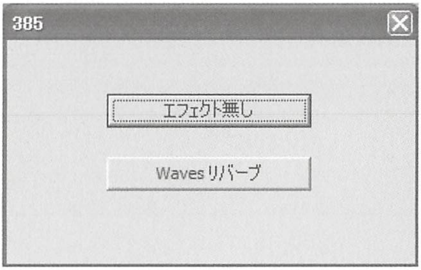
●サウンドにウェーブスリバーブエフェクトをかける

初級

サンプルプロジェクト　385　ウェーブス・リバーブ

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_WAVES_REVERBを指定して、SetFXメソッドを実行します。

▼実行結果



●ウェーブス・リバーブエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_WAVES_REVERB);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID*  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  *pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずウェーブス・リバーブエフェクトがかかります。

何も指定しないとデフォルトのウェーブス・リバーブがかかりますが、好きなようにウェーブス・リバーブの特徴を指定することができます。それについては次のTipsを参考にしてください。

386 ウェーブス・リバーブ効果を調整する

●IDirectSoundFXWavesReverb8:: SetAllParameters メソッドでウェーブス・リバーブ特性を設定する 初級

サンプルプロジェクト 385 ウェーブス・リバーブ

ウェーブス・リバーブ特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXWavesReverb8インターフェイスを取得することから始めます。

IDirectSoundFXWavesReverb8を取得したら、DSFXWavesReverb構造体を用意し、メンバを好きなように設定します。

DSFXWavesReverb構造体を初期化し終えたらIDirectSoundFXWavesReverb8:: SetAllParametersメソッドで設定し、完了です。

●ウェーブス・リバーブ特有のパラメーター設定

```
IDirectSoundFXWavesReverb8＊  pWavesReverb;
if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_WAVES_REVERB,
    0,IID_IDirectSoundFXWavesReverb8,(VOID＊＊)&pWavesReverb)))
{
    return  E_FAIL;
}
DSFXWavesReverb  WavesReverb;
//-96～0（dB）
WavesReverb.fInGain=DSFX_WAVESREVERB_INGAIN_DEFAULT;
///-96～0（dB）
WavesReverb.fReverbMix=DSFX_WAVESREVERB_REVERBMIX_DEFAULT  ;
//0.001～1000.0（ミリ秒）
WavesReverb.fReverbTime=DSFX_WAVESREVERB_REVERBTIME_DEFAULT  ;
//      0.001～0.999（比率）
WavesReverb.fHighFreqRTRatio=DSFX_WAVESREVERB_HIGHFREQRTRATIO_DEFAULT;

//エフェクトパラメーターをセット
if(FAILED(pWavesReverb->SetAllParameters(&WavesReverb)))
{
    pWavesReverb->Release();
    return  E_FAIL;
}
pWavesReverb->Release();
```

設定が終わったら、もはやIDirectSoundFXWavesReverb8に用はないのでリリースしましょう。

●パラメータ用の構造体

```
typedef  struct  _DSFXWavesReverb  {
    FLOAT    fInGain;
    FLOAT    fReverbMix;
    FLOAT    fReverbTime;
    FLOAT    fHighFreqRTRatio;
}  DSFXWavesReverb,  ＊LPDSFXWavesReverb;
```

●メンバの意味

fInGain

入力音の音量。最低にすると結果的に出力音も聞えなくなります。

fReverbMix

原音とリバーブ音の比率。

fReverbTime

リバーブ音のディレイ時間。

fHighFreqRTRatio

高周波数リバーブの時間比率。

I3DL2リバーブ

387 I3DL2リバーブとは？

●I3DL2団体のリバーブ

初級

リバーブについての概要はTips384のWavesリバーブで既に解説しています。DirectSoundのリバーブとしては、このI3DL2リバーブだけです。I3DL2とはI3DL2団体のリバーブでだからです。

I3DL2リバーブはパラメーターが多いのが特徴的ですが、プリセットが大量に用意されているので、それを利用すれば逆にすべてのエフェクトの中で最もパラメーター設定が楽なエフェクトと言えます。

388 I3DL2リバーブをかける

●サウンドにI3DL2リバーブエフェクトをかける

初級

サンプルプロジェクト　388　I3DL2リバーブ

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_I3DL2REVERBを指定して、SetFXメソッドを実行します。

I3DL2リバーブの場合は、他のエフェクトサンプルと若干異なります。

プリセットを利用するので、PlaySound関数の引数を一つ多くして、プリセットを受け渡しできるようにしています。

呼び出し元はプリセットの番号を渡してPlaySound関数をコールします。

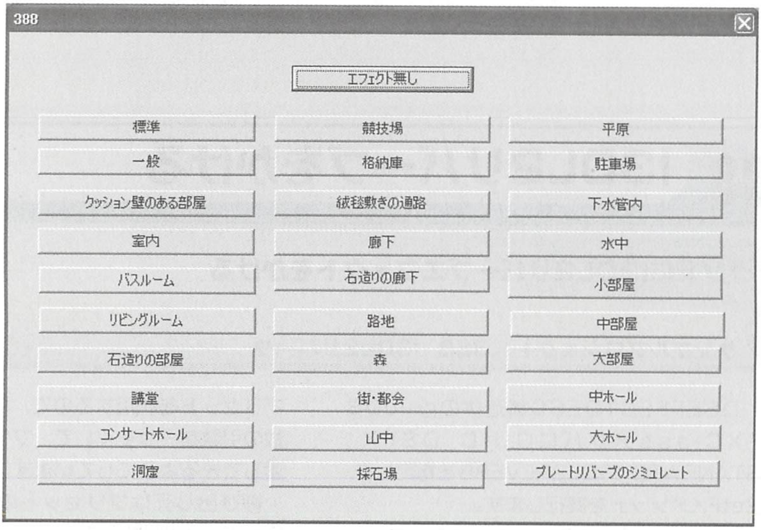
●I3DL2リバーブエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySounds(0,&GUID_DSFX_STANDARD_I3DL2REVERB,GetPreset(wParam));
```

```
HRESULT  PlaySounds(DWORD  dwSoundIndex,const  GUID＊  pGuid,DWORD  dwPreset)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();
    //エフェクトが指定されている場合は、そのエフェクトをバッファーにセットする
    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
```

```
memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
dsEffect.dwFlags  =  0;
dsEffect.guidDSFXClass  =  *pGuid;
if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))
{
    MessageBox(0,"¥nエフェクト設定失敗  ¥n","エラー",MB_OK);
    return  E_FAIL;
}
//再生する
g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

▼実行結果


388
エフェクト無し
標準
競技場
平原
一般
格納庫
駐車場
クッション壁のある部屋
絨毯敷きの通路
下水管内
室内
廊下
水中
バスルーム
石造りの廊下
小部屋
リビングルーム
路地
中部屋
石造りの部屋
森
大部屋
講堂
街・都会
中ホール
コンサートホール
山中
大ホール
洞窟
採石場
プレートリバーブのシミュレート

プリセットさえ指定すれば、他のエフェクトのようにそれ専用の構造体を初期化し、SetAllParametersメソッドを実行などという操作は不要になり、逆にコードは単純になります。

プリセットについては次のTipsを参考にしてください。

389 I3DL2リバーブ効果を調整する

●プリセットを利用してI3DL2リバーブ特性を設定する

初級

サンプルプロジェクト 388 I3DL2リバーブ

プリセット番号は本サンプルで勝手に決めたインデックスなので、それをDirectSoundの番号に変換します。それだけです。

```
DWORD  GetPreset(WPARAM  wParam)
{
    switch(wParam)
    {
        case  IDC_DEFAULT:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_DEFAULT;
        break;
        case  IDC_GENERIC:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_GENERIC;
        break;
        case  IDCPADDEDCELL:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_PADDEDCELL;
        break;
        case  IDC_ROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_ROOM;
        break;
        case  IDC_BATHROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_BATHROOM;
        break;
        case  IDC_LIVINGROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_LIVINGROOM;
        break;
        case  IDC_STONEROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_STONEROOM;
        break;
        case  IDC_AUDITORIUM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_AUDITORIUM;
        break;
        case  IDC_CONCERTHALL:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_CONCERTHALL;
        break;
        case  IDC_CAVE:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_CAVE;
            break;
            case   IDC_ARENA:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_ARENA;
            break;
            case   IDC_HANGAR:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_HANGAR;
            break;
            case   IDC_CARPETEDHALLWAY:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_CARPETEDHALLWAY;
            break;
            case   IDC_HALLWAY:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_HALLWAY;
            break;
            case   IDC_STONECORRIDOR:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_STONECORRIDOR;
            break;
            case   IDC_ALLEY:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_ALLEY;
            break;
            case   IDC_FOREST:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_FOREST;
            break;
            case   IDC_CITY:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_CITY;
            break;
            case   IDC_MOUNTAINS:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_MOUNTAINS;
            break;
            case   IDC_QUARRY:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_QUARRY;
            break;
            case   IDC_PLAIN:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_PLAIN;
            break;
            case   IDC_PARKINGLOT:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_PARKINGLOT;
            break;
            case   IDC_SEWERPIPE:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_SEWERPIPE;
            break;
            case   IDC_UNDERWATER:
            return   DSFX_I3DL2_ENVIRONMENT_PRESET_UNDERWATER;
            break;
```

```
        case  IDC_SMALLROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_SMALLROOM;
        break;
        case  IDC_MEDIUMROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_MEDIUMROOM;
        break;
        case  IDC_LARGEROOM:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_LARGEROOM;
        break;
        case  IDC_MEDIUMHALL:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_MEDIUMHALL;
        break;
        case  IDC_LARGEHALL:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_LARGEHALL;
        break;
        case  IDC_PRESET_PLATE:
        return  DSFX_I3DL2_ENVIRONMENT_PRESET_PLATE;
        break;
    }
    return  0;
}
/
```

●パラメータ用の構造体

```
typedef  struct  _DSFXI3DL2Reverb  {
    LONG    lRoom;
    LONG    lRoomHF;
    FLOAT  flRoomRolloffFactor;
    FLOAT  flDecayTime;
    FLOAT  flDecayHFRatio;
    LONG    lReflections;
    FLOAT  flReflectionsDelay;
    LONG    lReverb;
    FLOAT  flReverbDelay;
    FLOAT  flDiffusion;
    FLOAT  flDensity;
    FLOAT  flHFReference;
} DSFXI3DL2Reverb,  ＊LPDSFXI3DL2Reverb;
```

●プリセットを使う！

I3DL2リバーブにはプリセットが用意されているので大変便利になっています。プリセットは実に30種類もあり、十分すぎるほどの広範囲をカバーしています。

実際、本サンプルではプリセットを使用しているので、他のエフェクトサンプルとは違い、手作業でパラメーター・メンバ変数を初期化してはいません。

●I3DL2_ENVIRONMENT_PRESET_DEFAULT「デフォルト」

```
lRoom         -1000
lRoomHF       -100
flRoomRolloffFactor      0
flDecayTime          1.49
flDecayHFRatio       0.83
lReflections         -2602
flReflectionsDelay         0.007
lReverb      200
flReverbDelay        0.011
flDiffusion          100
flDensity  100
flHFReference        5000
```

●I3DL2_ENVIRONMENT_PRESET_GENERIC「一般」

```
lRoom         -1000
lRoomHF       -100
flRoomRolloffFactor      0
flDecayTime          1.49
flDecayHFRatio       0.83
lReflections         -2602
flReflectionsDelay         0.007
lReverb      200
flReverbDelay        0.011
flDiffusion          100
flDensity  100
flHFReference        5000
```

●I3DL2_ENVIRONMENT_PRESET_PADDEDCELL「クッション壁のある部屋」

```
lRoom         -1000
lRoomHF       -454
flRoomRolloffFactor      0
flDecayTime          0.4
flDecayHFRatio       0.83
lReflections         -1646
```

```
flReflectionsDelay      0.002
lReverb     53
flReverbDelay      0.003
flDiffusion        100
flDensity  100
flHFReference      5000
```

●I3DL2_ENVIRONMENT_PRESET_ROOM「室内」

```
lRoom         -1000
lRoomHF       -454
flRoomRolloffFactor      0
flDecayTime          0.4
flDecayHFRatio     0.83
lReflections         -1646
flReflectionsDelay      0.002
lReverb     53
flReverbDelay      0.003
flDiffusion        100
flDensity  100
flHFReference      5000
```

●I3DL2_ENVIRONMENT_PRESET_BATHROOM「バスルーム」

```
lRoom         -1000
lRoomHF       -1200
flRoomRolloffFactor      0
flDecayTime          1.49
flDecayHFRatio     0.54
lReflections         -370
flReflectionsDelay      0.007
lReverb     1030
flReverbDelay      0.011
flDiffusion        100
flDensity  60
flHFReference      5000
```

●I3DL2_ENVIRONMENT_PRESET_LIVINGROOM「リビングルーム」

```
lRoom         -1000
lRoomHF       -6000
flRoomRolloffFactor      0
flDecayTime          0.5
flDecayHFRatio     0.1
lReflections         -1376
```

flReflectionsDelay	0.003
lReverb	-1104
flReverbDelay	0.004
flDiffusion	100
flDensity	100
flHFReference	5000

●I3DL2_ENVIRONMENT_PRESET_STONEROOM「石造りの部屋」

lRoom	-1000
lRoomHF	-300
flRoomRolloffFactor	0
flDecayTime	2.3
flDecayHFRatio	0.64
lReflections	-711
flReflectionsDelay	0.012
lReverb	83
flReverbDelay	0.017
flDiffusion	100
flDensity	100
flHFReference	5000

●I3DL2_ENVIRONMENT_PRESET_AUDITORIUM「講堂」

lRoom	-1000
lRoomHF	-476
flRoomRolloffFactor	0
flDecayTime	4.32
flDecayHFRatio	0.59
lReflections	-789
flReflectionsDelay	0.02
lReverb	-289
flReverbDelay	0.03
flDiffusion	100
flDensity	100
flHFReference	5000

●I3DL2_ENVIRONMENT_PRESET_CONCERTHALL「コンサートホール」

lRoom	-1000
lRoomHF	-500
flRoomRolloffFactor	0
flDecayTime	3.92
flDecayHFRatio	0.7
lReflections	-1230

```
flReflectionsDelay      0.02
lReverb     -2
flReverbDelay       0.029
flDiffusion         100
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_CAVE「洞窟」

```
lRoom       -1000
lRoomHF     0
flRoomRolloffFactor     0
flDecayTime         2.91
flDecayHFRatio      1.3
lReflections        -602
flReflectionsDelay      0.015
lReverb     -302
flReverbDelay       0.022
flDiffusion         100
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_ARENA「競技場」

```
lRoom       -1000
lRoomHF     -698
flRoomRolloffFactor     0
flDecayTime         7.24
flDecayHFRatio      0.33
lReflections        -1166
flReflectionsDelay      0.02
lReverb     16
flReverbDelay       0.03
flDiffusion         100
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_HANGAR「格納庫」

```
lRoom       -1000
lRoomHF     -1000
flRoomRolloffFactor     0
flDecayTime         10.05
flDecayHFRatio      0.23
lReflections        -602
```

```
flReflectionsDelay      0.02
lReverb    198
flReverbDelay     0.03
flDiffusion       100
flDensity  100
flHFReference     5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_CARPETEDHALLWAY「絨毯敷きの通路」

```
lRoom        -1000
lRoomHF     -4000
flRoomRolloffFactor      0
flDecayTime        0.3
flDecayHFRatio     0.1
lReflections       -1831
flReflectionsDelay      0.002
lReverb     -1630
flReverbDelay     0.03
flDiffusion       100
flDensity  100
flHFReference     5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_HALLWAY「通路・廊下」

```
lRoom        -1000
lRoomHF     -300
flRoomRolloffFactor      0
flDecayTime        1.49
flDecayHFRatio     0.59
lReflections       -1219
flReflectionsDelay      0.007
lReverb     441
flReverbDelay     0.011
flDiffusion       100
flDensity  100
flHFReference     5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_STONECORRIDOR「石造りの廊下」

```
flReflectionsDelay      0.013
lReverb     395
flReverbDelay     0.02
flDiffusion       100
flDensity  100
flHFReference     5000
```

```
lRoom       -1000
lRoomHF     -237
flRoomRolloffFactor     0
flDecayTime         2.7
flDecayHFRatio      0.79
lReflections        -1214
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_ALLEY「路地」

```
lRoom       -1000
lRoomHF     -270
flRoomRolloffFactor     0
flDecayTime         1.49
flDecayHFRatio      0.86
lReflections        -1204
flReflectionsDelay      0.007
lReverb     -4
flReverbDelay       0.011
flDiffusion         100
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_FOREST「森」

```
lRoom       -1000
lRoomHF     -3300
flRoomRolloffFactor     0
flDecayTime         1.49
flDecayHFRatio      0.54
lReflections        -2560
flReflectionsDelay      0.162
lReverb     -613
flReverbDelay       0.088
flDiffusion         79
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_CITY「街・都会」

```
lRoom       -1000
lRoomHF     -800
flRoomRolloffFactor     0
flDecayTime         1.49
flDecayHFRatio      0.67
```

```
lReflections        -2273
flReflectionsDelay         0.007
lReverb     -2217
flReverbDelay       0.011
flDiffusion         50
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_MOUNTAINS「山中」

```
lRoom         -1000
lRoomHF       -2500
flRoomRolloffFactor        0
flDecayTime         1.49
flDecayHFRatio      0.21
lReflections        -2780
flReflectionsDelay         0.3
lReverb     -2014
flReverbDelay       0.1
flDiffusion         27
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_QUARRY「採石場」

```
lRoom         -1000
lRoomHF       -1000
flRoomRolloffFactor        0
flDecayTime         1.49
flDecayHFRatio      0.83
lReflections        -10000
flReflectionsDelay         0.061
lReverb     500
flReverbDelay       0.025
flDiffusion         100
flDensity  100
flHFReference       5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_PLAIN「平原」

```
lRoom         -1000
lRoomHF       -2000
flRoomRolloffFactor        0
flDecayTime         1.49
flDecayHFRatio      0.5
```

lReflections	-2466
flReflectionsDelay	0.179
lReverb	-2514
flReverbDelay	0.1
flDiffusion	21
flDensity	100
flHFReference	5000

●DSFX_I3DL2_ENVIRONMENT_PRESET_PARKINGLOT「駐車場」

lRoom	-1000
lRoomHF	0
flRoomRolloffFactor	0
flDecayTime	1.65
flDecayHFRatio	1.5
lReflections	-1363
flReflectionsDelay	0.008
lReverb	-1153
flReverbDelay	0.012
flDiffusion	100
flDensity	100
flHFReference	5000

●DSFX_I3DL2_ENVIRONMENT_PRESET_SEWERPIPE「下水管内」

lRoom	-1000
lRoomHF	-1000
flRoomRolloffFactor	0
flDecayTime	2.81
flDecayHFRatio	0.14
lReflections	429
flReflectionsDelay	0.014
lReverb	648
flReverbDelay	0.021
flDiffusion	80
flDensity	60
flHFReference	5000

●DSFX_I3DL2_ENVIRONMENT_PRESET_UNDERWATER「水中」

lRoom	-1000
lRoomHF	-4000
flRoomRolloffFactor	0
flDecayTime	1.49
flDecayHFRatio	0.1

```
lReflections     -449
flReflectionsDelay     0.007
lReverb   1700
flReverbDelay    0.011
flDiffusion      100
flDensity 100
flHFReference    500
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_SMALLROOM「小部屋」

```
lRoom     -1000
lRoomHF   -600
flRoomRolloffFactor    0
flDecayTime      1.1
flDecayHFRatio   0.83
lReflections     -400
flReflectionsDelay     0.005
lReverb   500
flReverbDelay    0.01
flDiffusion      100
flDensity 100
flHFReference    5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_MEDIUMROOM「中部屋」

```
lRoom     -1000
lRoomHF   -600
flRoomRolloffFactor    0
flDecayTime      1.3
flDecayHFRatio   0.83
lReflections     -1000
flReflectionsDelay     0.01
lReverb   -200
flReverbDelay    0.02
flDiffusion      100
flDensity 100
flHFReference    5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_LARGEROOM「大部屋」

```
lRoom     -1000
lRoomHF   -600
flRoomRolloffFactor    0
flDecayTime      1.5
flDecayHFRatio   0.83
```

```
lReflections      -1600
flReflectionsDelay       0.02
lReverb     -1000
flReverbDelay      0.04
flDiffusion        100
flDensity  100
flHFReference      5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_MEDIUMHALL「中ホール」

```
lRoom         -1000
lRoomHF      -600
flRoomRolloffFactor        0
flDecayTime         1.8
flDecayHFRatio      0.7
lReflections       -1300
flReflectionsDelay       0.015
lReverb      -800
flReverbDelay      0.03
flDiffusion        100
flDensity  100
flHFReference      5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_LARGEHALL「大ホール」

```
lRoom         -1000
lRoomHF      -600
flRoomRolloffFactor        0
flDecayTime         1.8
flDecayHFRatio      0.7
lReflections       -2000
flReflectionsDelay       0.03
lReverb      -1400
flReverbDelay      0.06
flDiffusion        100
flDensity  100
flHFReference      5000
```

●DSFX_I3DL2_ENVIRONMENT_PRESET_PLATE「プレートリバーブのシミュレート」

```
lRoom         -1000
lRoomHF      -200
flRoomRolloffFactor        0
flDecayTime         1.3
flDecayHFRatio      0.9
```

```
lReflections      0
flReflectionsDelay      0.002
lReverb    0
flReverbDelay      0.01
flDiffusion        100
flDensity  75
flHFReference      5000
```

パラメトリック・イコライザー

390 パラメトリック・イコライザーとは？

●周波数の指定された範囲の倍音をブーストあるいはカットする

初級

パラメトリック・イコライザーといちいち発音するのは長いので、大抵は略してパライコと呼ばれます。このパライコは、指定した中心周波数の前後の周波数（この幅も指定できる）帯域の倍音をブーストあるいはカットする、ピーキングタイプのパライコです。通常のパライコは3つや多いもので4つ程度のバンドがありますが、DirectXのものは1つしかありません。

パライコの機能および目的は、指定された周波数の指定された範囲（帯域）の倍音（周波数、音程）をブーストあるいはカットすることです。

使用例としては、例えば、原音の高音部分を強調したいときは、バンドを高周波数帯域にセットしてゲイン（音量）を上げます。ここでのサンプルもそのような使用方法をとっています。

391 パラメトリック・イコライザーをかける

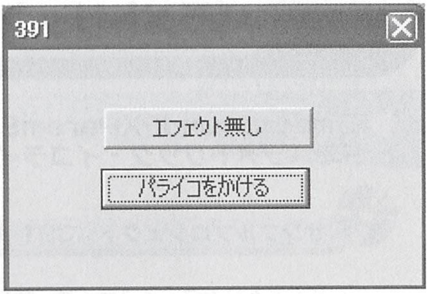
●サウンドにパラメトリックイコライザーエフェクトをかける

初級

サンプルプロジェクト　391　パライコ

DSEFFECTDESC構造体のguidDSFXClassメンバにGUID_DSFX_STANDARD_PARAMEQを指定して、SetFXメソッドを実行します。

▼実行結果



●パラメトリック・イコライザーエフェクトのセット

```
case  IDC_BUTTON2:
    PlaySound(0,&  GUID_DSFX_STANDARD_PARAMEQ);
```

```
HRESULT  PlaySound(DWORD  dwSoundIndex,const  GUID*  pGuid)
{
    g_pDSBuffer[  dwSoundIndex  ]->Stop();

    DSEFFECTDESC  dsEffect;
    DWORD  dwResult=NULL;
    memset(&dsEffect,  0,  sizeof(DSEFFECTDESC));
    dsEffect.dwSize  =  sizeof(DSEFFECTDESC);
    dsEffect.dwFlags  =  0;
    dsEffect.guidDSFXClass  =  *pGuid;
    if(FAILED(g_pDSBuffer[  dwSoundIndex  ]->SetFX(1,&dsEffect,&dwResult)))

    g_pDSBuffer[  dwSoundIndex  ]->Play(NULL,0,NULL);
```

これでとりあえずパラメトリック・イコライザーエフェクトがかかります。

何も指定しないとデフォルトのパラメトリック・イコライザーがかかりますが、好きなようにパラメトリック・イコライザーの特徴を指定することができます。それについては、次のTipsを参考にしてください。

392 パラメトリック・イコライザー効果を調整する

●IDirectSoundFXParamEq8:: SetAllParametersメソッドでパラメトリック・イコライザー特性を設定する

初級

サンプルプロジェクト　391　パライコ

パラメトリック・イコライザー特有のパラメーターは、まずセカンダリバッファ内からIDirectSoundFXParamEq8インターフェイスを取得することから始めます。

IDirectSoundFXParamEq8を取得したら、DSFXParamEq構造体を用意し、メンバを好きなように設定します。

DSFXParamEq構造体を初期化し終えたらIDirectSoundFXParamEq8::SetAllParametersメソッドで設定し、完了です。

●パラメトリック・イコライザー特有のパラメーター設定

```
IDirectSoundFXParamEq8＊  pParamEq;
    if(FAILED(pDSBuffer8->GetObjectInPath(GUID_DSFX_STANDARD_PARAMEQ,
           0,IID_IDirectSoundFXParamEq8,(VOID＊＊)&pParamEq)))
    {
           return  E_FAIL;
    }
    DSFXParamEq  dsParamEq;
    //80～16000（Hz）
    dsParamEq.fCenter=12000;
    //1～36（周波数帯域（倍音構成）、半音単位）
    dsParamEq.fBandwidth=20;
    //-15～15（dB）
    dsParamEq.fGain=15;
    //エフェクトパラメーターをセット
    if(FAILED(pParamEq->SetAllParameters(&dsParamEq)))
    {
           pParamEq->Release();
           return  E_FAIL;
    }
    pParamEq->Release();
```

設定が終わったら、もはやIDirectSoundFXParamEq8に用はないのでリリースしましょう。

●パラメータ用の構造体

```
typedef  struct  _DSFXParamEq  {
    FLOAT    fCenter;
    FLOAT    fBandwidth;
    FLOAT    fGain;
} DSFXParamEq,  *LPDSFXParamEq;
```

●メンバの意味

ダイアルが一個だけのパライコを想像してください。

fCenter

調整したい音程の中心周波数

fBandwidth

fCenterを中心とする、調整したい音程の範囲

fGain

上2つのパラメーターで指定した帯域の音量。上げればブースト、下げればカットすることになる。

393 3Dサウンドとは？

●現実世界での音の聞え方をシミュレートするもの

初級

3Dサウンドとは、現実世界での音の聞え方をシミュレートするものです。例えば、ヘリコプターが本当に自分の周りをぐるぐる飛行しているかのように聞こえさせたり、ゲーム内で後ろから声を掛けられるシーンでは、後ろから聞こえているように出力するものです。

DirectSoundの3Dサウンド部分はDirectSound3Dと呼ばれます。DirectSound3Dの実体はDirectSoundコンポーネント内にあるIDirectSound3DBuffer8インターフェイスとIDirectSound3DListener8インターフェイスです。

3Dサウンドの効果を実現する最も簡単で有効な手段は、視聴者を取り囲むようにスピーカーを配置することです。最近ではPCスピーカーを5.1チャンネル環境にする人も増えてきていると思いますが、そのような環境は3D環境を簡単に実現できうる環境とも言えます。実際、DirectSound3Dをマルチスピーカー環境で聞くことが最も効果的です。

しかし、一般的にはまだまだ2スピーカーの環境が圧倒的に多いと思います。一般的なステレオ環境（2スピーカー）では前左、前右にスピーカーを配置していると思いますが、そのようなステレオ環境では通常、後方からの音を再現できません。一方、マルチスピーカーの場合は何も難しいことを考えずに、後方の音は後ろのスピーカーから出せばいいだけの話なので、技術的には各スピーカーの音量調整くらいしか気にすることはありません。ところが、2スピーカーの場合は、物理的に

後ろから音を出すことは不可能なので、なんらかのトリックが必要になってきます。一般的に2スピーカーで3Dサウンドを実現する技術は“仮想（バーチャル）3Dサウンド”と呼ばれていますが、DirectSound3Dの本来の目的はこの2スピーカーでの仮想3Dサウンドです。もちろんDirectSound3Dはマルチスピーカー環境でも必要であり、マルチスピーカーの場合のほうがリアルな3Dサウンドになります。

394 HRTFとは？

●HRTFとはHead-Related Transfer Function：頭部伝達関数の意

初級

DirectSound3Dは、仮想3Dサウンドをよりリアルにするために HRTF（Head-Related Transfer Function：頭部伝達関数）という技術を採用しています。HRTFはいろいろな研究機関で研究され、それぞれの団体でその定義は微妙に異なり、精度はまちまちです。DirectSound3DにおけるHRTFの概要は次のようになります。

人間の2つの耳は左右に離れているので（当たり前ですが）、一方方向からくる音は、もう片方の耳に届くまで極僅かに遅れます。DirectSound3Dは、この時間的ズレを1ミリ秒としています。

正面からくる音は、まず鼻に反射して変化し、さらに頬骨によっても反射します。

後方からくる音は後頭部で反射してさら耳の裏でも反射します。HRTFは、驚くことに“耳たぶ“や”耳のひだ“による微妙な反射をも考慮しています。

HRTFはこのように、非常に細かい部分での音変化を計算して、仮想3Dサウンドを実現します。最近では、携帯型mp3プレイヤーでHRTFによる仮想3Dサウンドを実装しているものを見かけます。

395 セカンダリバッファに3D効果をかける

●3D再生を行う

中級

サンプルプロジェクト　395　3D再生

4つのスピーカーから独立した音を出せる環境の場合は確実に3D効果を体感できます。

一般的な2スピーカーの場合は、定位を気にする必要があります。3D効果が分からない場合は自分自身が動いて（笑）3Dに聞えるようになるような場所（定位）を探してください。必ず3Dに聞える場所はあるはずです。

このサンプルは、ヘリコプターのプロペラ音を、リスナーを中心とする円周上で回転させるものです。最低4つのスピーカーがあり、

なおかつ各スピーカーから独立した音を出せるような出力ラインがある環境であれば、かなりリアルに聞えます。また、2スピーカーであってもHRTFにより仮想3D効果がかかるので、それなりに聞えます。実は筆者が驚いたのは2スピーカー環境でも後ろから音が聞えるような錯覚を起こしたことです。

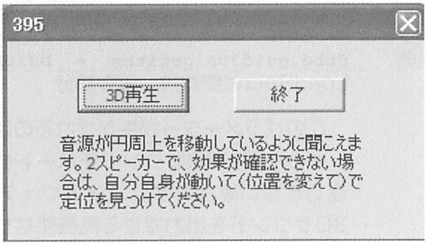
#include <d3dx9.h>

3D空間内における音源の回転を計算する際にはベクトルと行列を使用していますが、Direct3Dのほうで便利な算術関数とデータ型が用意されているのでそれを使用しない理由はありません。Direct3Dコンポーネントのヘッダーをインクルードしているのはそのためです。Direct3Dと言っても使用しているのはDirect3Dエクステンションの算術関数だけです。

case WM_TIMER:ブロック

Win32タイマーメッセージにより、このブロックに入ります。このサンプルは200ミリ秒ごとに音源の位置を更新しています。

▼実行結果



音源が円周上を移動しているように聞こえます。2スピーカーで、効果が確認できない場合は、自分自身が動いて(位置を変えて)で定位を見つけてください。

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

実際にDirectSoundに位置をセットしているのはこの行です。引数として3次元ベクトルのxyz成分を渡します。

```
fAngle+=D3DXToRadian(0.1);
```

200ミリ秒ごとに0.1度回転させています。算術関数の引数の関係で度数をラジアンに変換しています。

```
D3DXMatrixRotationY(&matRotation,fAngle);
```

現在の回転角を元に回転行列を作成します。

```
D3DXVec3TransformCoord(&vecHeli,&vecHeli,&matRotation);
```

その回転行列により、音源の位置ベクトルvecHeliを座標変換（回転）させます。

case IDC_BUTTON1:

IDC_BUTTONはダイアログの“再生”ボタンのIDです。

ユーザーが再生ボタンを押したときに、このブロックに入ります。処理としては、再生のための準備処理です

PlaySound(0,NULL);

まず、サウンドバッファを停止します。3D関係の設定をするときはバッファが停止していなくてはなりません。

```
SetTimer(hwndDlg,1,200,NULL);//0.2秒間隔のタイマーを設定
```

200ミリ秒間隔でタイマーメッセージを発行するように設定します。

```
vecHeli=D3DXVECTOR3(0,20,10);
```

ヘリ（音源）の位置を前方20メートル、上10メートルに初期化します。

fAngle=0;

回転角をゼロにします。

```
dsbd.guid3DAlgorithm  =  DS3DALG_HRTF_FULL;
```

LoadSound関数内、固有部分

このパラメーターが生かされるのは、サウンドカードが3Dサウンドをハードウェア処理できない場合です。ハードウェアにより3Dサウンドを出力できる高機能なサウンドカードでは、意味がありません。

サウンドカードがハードウェア処理できない場合、DirectSoundは、3Dミキシングをソフト的にエミュレートしますが、これはその品質を指定します。ここでは、最高品質であるDS3DALG_HRTF_FULLを指定しています。

ハードウェアミキシング能力のあるサウンドカードの場合、ここでDS3DALG_NO_VIRTUALIZATIONやDS3DALG_DEFAULT　GUID_NULLを指定しても、3Dアルゴリズムが適用されます。

```
g_pDS3DBuffer[  dwSoundIndex  ]->SetMinDistance(15,DS3D_IMMEDIATE);
g_pDS3DBuffer[  dwSoundIndex  ]->SetMaxDistance(20,DS3D_IMMEDIATE);
```

最小距離と最大距離の設定をしています。最小距離とは、それ以上近づいても音が大きくならない距離のことであり、最大距離とは、それ以上離れても音が小さくならない距離のことです。

```
g_pDS3DBuffer[  dwSoundIndex  ]->SetVelocity(0,0,40,DS3D_IMMEDIATE);
```

音源の速度を設定します。速度といってもこれはドップラー効果の計算で使われる速度という意味です。回転の速度等ではありません。

```
LPDIRECTSOUNDBUFFER            pPrimary;
ZeroMemory(&dsbd,  sizeof(DSBUFFERDESC));
dsbd.dwSize  =  sizeof(DSBUFFERDESC);
dsbd.dwFlags  =  DSBCAPS_CTRL3D  |  DSBCAPS_PRIMARYBUFFER;
if  (SUCCEEDED(g_pDSound->CreateSoundBuffer(&dsbd,  &pPrimary,  NULL)))
{
            pPrimary->QueryInterface(IID_IDirectSound3DListener8,(LPVOID
*)&g_p3DListener);
```

```
    pPrimary->Release();
}
```

なぜここでプライマリーバッファを作成しているかというと、リスナーインターフェイスを取得するためです。ここでは、その目的のためだけにプライマリーバッファを一時的に作成・使用しています。

```
g_p3DListener->SetPosition(0,0,0,DS3D_IMMEDIATE);
```

リスナーの位置は、デフォルトでxyz成分がゼロですが、念のため初期化しておきます。

```
g_p3DListener->SetDopplerFactor(100,DS3D_IMMEDIATE);
```

ドップラー効果全体の“かかり具合”を100に設定しています

396 音源を空間的に移動する

●IDirectSound3Dbuffer:: SetPositionメソッドにより音源の位置を更新する 中級

サンプルプロジェクト 396 音源移動

音源の位置はIDirectSound3Dbuffer::SetPositionメソッドで変更でき、それにより聞え方を変化させます。

SetPositionメソッドには3つのFLOAT値を渡しますが、それぞれは3D空間のx,y,z座標を意味します。

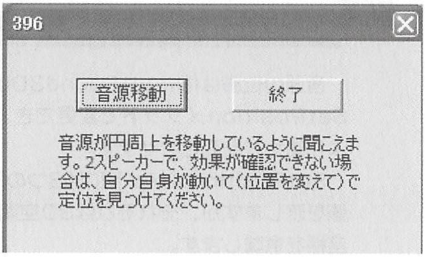
サンプルでは、D3DX系の関数を利用していますが、自分で3D座標を計算することができれば、特にDirect3Dの力を借りることはありません。

サンプルは単純なダイアログスタイルなので、毎フレームの処理を継続するために0.2秒間隔でタイマー通知を発行するようにしています。

0.2秒ごとにおこなっている処理は、音源の位置更新で次のような単純なものです。

まず、音源をヘリコプターに見立てているので、ベクトルの変数名はvexHeliとしています。D3DXVECTOR3型で宣言しています。

▼実行結果



音源が円周上を移動しているように聞こえます。2スピーカーで、効果が確認できない場合は、自分自身が動いて(位置を変えて)で定位を見つけてください。

```
static  D3DXVECTOR3  vecHeli(0,20,10);
```

音源の座標を毎フレーム増減させるための増分変数をiDeltaとして用意し、初期値はプラス1です。

```
static  INT  iDelta=1;
```

座標を増減させます。適当にプラスマイナス15単位に達したら、また増分変数の符号を反転させ、音源が継続的に往復するようにしています。

```
vecHeli.z+=iDelta;

if(vecHeli.z>=15  )
{
    iDelta=-1;
}
if(vecHeli.z<=-15)
{
    iDelta=1;
}
```

更新された座標を毎フレームごとに3Dバッファにセットすれば、DirectSound3Dが変化を反映してくれます。

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

397 音源を空間的に回転させる

●IDirectSound3Dbuffer:: SetPositionメソッドにより音源の位置を更新する　中級

サンプルプロジェクト　397　3D音源回転

音源の位置はIDirectSound3Dbuffer::SetPositionメソッドで変更でき、それにより聞え方を変化させます。

SetPositionメソッドには3つのFLOAT値を渡しますが、それぞれは3D空間のx,y,z座標を意味します。

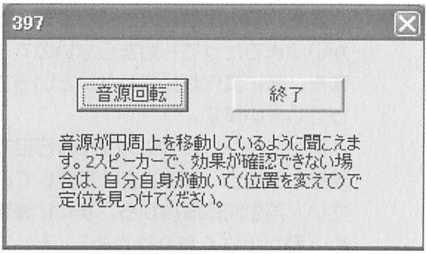
サンプルでは、D3DX系の関数を利用していますが、自分で3D座標を計算することができれば、特にDirect3Dの力を借りることはありません。

サンプルは単純なダイアログスタイルなので、毎フレームの処理を継続するために0.2

秒間隔でタイマー通知を発行するようにしています。0.2秒ごとにおこなっている処理は、音源の位置更新で次のような単純なものです。

まず、音源をヘリコプターに見立てているので、ベクトルの変数名はvexHeliとしています。D3DXVECTOR3型で宣言しています。

▼実行結果



音源が円周上を移動しているように聞こえます。2スピーカーで、効果が確認できない場合は、自分自身が動いて(位置を変えて)で定位を見つけてください。

```
static  D3DXVECTOR3  vecHeli(0,20,10);
```

座標を、ワールド原点を中心として回転させています。

```
static  FLOAT  fAngle=0;
D3DXMATRIX  matRotation;
fAngle+=D3DXToRadian(0.1);
D3DXMatrixRotationY(&matRotation,fAngle);
D3DXVec3TransformCoord(&vecHeli,&vecHeli,&matRotation);
```

更新された座標を毎フレームごとに3Dバッファにセットすれば、DirectSound3Dが変化を反映してくれます。

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

398 音源の最小・最大距離を設定する

最大・最小距離を理解する

中級

サンプルプロジェクト　395 3D再生　396 音源移動　397 3D音源回転

最小距離と最大距離は、勘違いしやすい概念です。勘違いとは“逆の意味”で捉えてしまうことです。最小距離とは、その地点を越えると音が小さくなり始める地点までの距離です。最大距離とは、その地点を越えると音の減衰を行わない（音を小さくしない）地点までの距離です。

したがって、最大距離を小さくすると音はいつまでも鳴り続け、あまりにも小さいと減衰すら起こらないので、音に変化がまったく

無くなってしまいます。

また、最小距離を大きくしすぎると、減衰がいつまでたっても始まらないので、これもまた、音に変化が見られないということになってしまいます。

試しにサンプル396の238行目で最小距離を15など、少ない値に変更してみてください。再生開始直後から、すぐに減衰しはじめ（音が小さくなりはじめ）、そして、音源の増分変数iDeltaが反転する地点（+−150単位）付近では音が聞こえなくなります。

なお、サンプル396で最小距離を150にしているのは、左右での音の移動だけを強調するために減衰を行わせたくなかったからです。音源は[−150,150]の間で移動しますから、最小距離を150にすれば減衰が起こらないというわけです。

最小・最大距離のセットはIDirectSound3Dbufferインターフェイスのメソッドで行います。

IDirectSound3Dbuffer:: SetMinDistance及びSetMaxDistanceメソッド

サンプルでのコードは次のようになっています。

```
g_pDS3DBuffer[  dwSoundIndex  ]->SetMinDistance(150,DS3D_IMMEDIATE);
g_pDS3DBuffer[  dwSoundIndex  ]->SetMaxDistance(1000,DS3D_IMMEDIATE);
```

399 音源の速度を設定する

IDirectSound3Dbuffer:: SetVelocityメソッドで速度をセットする　中級

サンプルプロジェクト　395　3D再生　396　音源移動　397　3D音源回転

音源にはスピードをセットすることができ、IDirectSound3Dbuffer:: SetVelocityメソッドでセットします。

しかし、サンプルではアプリケーション側で位置を毎回更新しているわけですから、音源自体がスピードを持っている必要はありません。というよりも、そもそもSetVelocityメソッドでスピードを設定したとしても実は無意味です。

スピードを指定する目的は“ドップラー効果”のパラメーターとして渡しているに過ぎません。音の減衰に影響することはありません。

サンプルでのコードは次のとおりです。

```
//音源の速度を設定しておく。時速40キロメートル
g_pDS3DBuffer[  dwSoundIndex  ]->SetVelocity(0,0,40,DS3D_IMMEDIATE);
```

サンプルではリスナーインターフェイス側でドップラー効果をセットしているので、音源側でもスピードをセットしたままでです。

400 リスナーとは？

DirectSound3Dでのリスナーオブジェクト　初級

リスナーとは呼んで字の如く、“聞き手”(現実にPCの前にいる人、あなたかしれないし、他の人かもしれませんが) のオブジェクトです。

音源を動かしても、リスナーを動かしても、どちらでも両者の相対的な位置や速度を変えることなので、結果として音に与える影響は同じなので、リスナーオブジェクトの意義が薄く感じるかも知れませんが、リスナーと音源が別々のオブジェクトになっているほうが、音場をモデリングしやすいでしょう。

なお、リスナーインターフェイスはドップラー効果を持っています。

401 リスナーを空間的に移動する

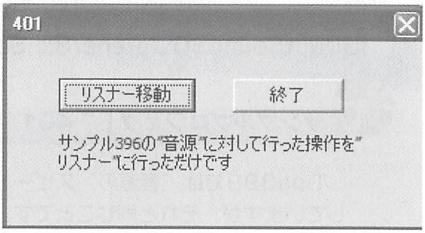
IDirectSound3DListener8:: SetPositionメソッドにより音源の位置を更新する　中級

サンプルプロジェクト　401 リスナー移動

このサンプルのポイントは非常に単純です。

サンプル396における、座標の更新対象を“音源”から“リスナー”にしただけです。コード上はたった1行変更しただけです。

▼実行結果



サンプル396の"音源"に対して行った操作を"リスナー"に行っただけです

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

```
g_p3DListener->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

もっと言えば、一行というより、単にインターフェイスを変えただけなのが分かるでしょう。

402 リスナーを空間的に回転させる

IDirectSound3DListener8:: SetPositionメソッドにより音源の位置を更新する　中級

サンプルプロジェクト　402　リスナー回転

このサンプルのポイントは、非常に単純です。

サンプル397における、座標の更新対象を“音源”から“リスナー”にしただけです。コード上の65行目を、たった1行変更しただけです。

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

```
g_p3DListener->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

もっと言えば、一行というより、単にインターフェイスを変えただけなのが分かるでしょう。

403 リスナーの速度を設定する

IDirectSound3DListener8:: SetPositionメソッドにより音源の位置を更新する　中級

サンプルプロジェクト　401　リスナー移動

Tips399では“音源の”スピードを設定していますが、それと同じことです。

音源のスピードはIDirectSound3Dbuffer:: SetVelocityでセットしましたが、リスナーのスピードは当然リスナーインターフェイスなのでIDirectSound3DListener8:: SetVelocityメソッドでセットします。

さて、音源とリスナーという2つの主体をモデリングしたインターフェイスがあるのは便利なのですが、同じようなメドッドがあり、そして、それらの効果が相対的に同じであるので少々混乱するかもしれません。

例えば次の3つの状態があるとします。

①音源が時速60kmでリスナーに近づき、リスナーは停止している。
②音源は停止していて、リスナーが時速60kmで音源に近づいている。
③音源とリスナーがともに時速30kmでお互いに近づいている。

これら3つは、どれも同じ結果になります。

404 リスナーにドップラー効果をかける

IDirectSound3DListener8:: SetDopplerFactorメソッドでドップラー効果をセットする 中級

サンプルプロジェクト 401 リスナー移動

ドップラー効果とは、例えば、救急車のサイレンがどのように聞えるか思い出してみてください。遠くで鳴っているときと近くで鳴っている時に音量が増減するのはもちろんですが、救急車がこちらに向かっている時と遠ざかっている時とでは、その音程も異なるでしょう。F1レース等で車が向かってくる時と、離れている時のエンジン音の聞こえ方も同様です。

この音程が変化する現象がドップラー効果と呼ばれるものです。音源とリスナーの相対速度が変化することにより、音の周波数が増減することが原因です。

ドップラー効果はリスナーインターフェイスのみが持つもので、IDirectSound3DListener8:: SetDopplerFactorによりセットします。

サンプルでのコードは次のとおりです。

```
g_p3DListener->SetDopplerFactor(100,DS3D_IMMEDIATE);
```

第1引数は、ドップラー係数を意味し、範囲はDS3D_MINDOPPLERFACTORからDS3D_MAXDOPPLERFACTORまでです。

デフォルトではDS3D_DEFAULTDOPPLERFACTORとなります。それぞれの定数は次のようにDsound.h内で定義されています。

```
#define  DS3D_MINROLLOFFFACTOR            0.0f
#define  DS3D_MAXROLLOFFFACTOR            10.0f
#define  DS3D_DEFAULTROLLOFFFACTOR     1.0f
```

音源の座標を毎フレーム増減させるための増分変数をiDeltaとして用意し、初期値はプラス1です。

```
static  INT  iDelta=1;
```

座標を増減させます。適当にプラスマイナス15単位に達したら、また増分変数の符号を反転させ、音源が継続的に往復するようにしています。

```
vecHeli.z+=iDelta;

if(vecHeli.z>=15  )
{
    iDelta=-1;
}
if(vecHeli.z<=-15)
{
    iDelta=1;
}
```

更新された座標を毎フレームごとに3Dバッファにセットすれば、DirectSound3Dが変化を反映してくれます。

```
g_pDS3DBuffer[0]->SetPosition(vecHeli.x,vecHeli.y,vecHeli.z,DS3D_IMMEDIATE);
```

405 マルチチャンネルとは？

●3チャンネル以上のチャンネル数を持つ

初級

モノラルとステレオの意味はご存知とは思いますが、念のため説明すると、モノラルは1チャンネルなので左右のスピーカーからは全く同じ音がでます（同じ音しか出ません）。一方ステレオは2チャンネルで、左右のスピーカーから異なる音を出してモノラルよりも臨場感を出すことができます。

マルチチャンネルとはチャンネル数が3つであることを意味します。したがってマルチチャンネルWAVとは、3チャンネル以上のチャンネル数を持つWAVEファイルです。

そして、マルチ再生とは3個以上のマルチスピーカー環境で各スピーカーから独立した音を出すことです。

マルチ再生に関しては、マイクロソフトのドキュメントにも書かれていません。

406

マルチチャンネルのハード環境を整える

スピーカーやサウンドカード出力ラインに注意する

初級

本書マルチサウンド関連のサンプルの動作確認は、マルチスピーカー環境である必要があります。当然ですが一般的な2スピーカーでは確認できません。

なお、映画DVDを鑑賞するために5.1チャンネル環境である読者も少なからずいると思いますが、その場合注意が必要です。

スピーカーが6つあるいはそれ以上あるからといって、生WAV（リニアPCM）のマルチ再生が出来るとは言いきれません。特に最安価格帯のスピーカーセットは映画ソフトの出力に特化しているものが多く、出力ラインが一本しかないものがほとんどです（出力ラインとスピーカーの数は別です）。

その場合、スピーカーから異なるリニアPCM音を出すことは不可能です。映画DVDを再生する分には各スピーカーから独立した音が出ますが、それはドルビーデジタルやdtsという圧縮フォーマット専用のデコーダーにより、1本のラインを流れる音を6分割して実現しているに過ぎないからです。

エンコード・デコード無しに各スピーカーから独立したPCMを出力するためには、スピーカーごとに出力ラインが必要です。

また、サウンドカード側も独立した出力ラインを持っている物である必要があります。サウンドカードやスピーカーセットを購入する場合は、「出力が何本あるか」ということを確認するようにしてください。

“ドルビーデジタル、dtsに対応！”とか“5.1チャンネル、7.1チャンネル”という謳い文句に惑われないようしましょう。

407

マルチチャンネルWAVファイルをマルチ再生する

4チャンネルのWAVファイルを4チャンネルで再生する

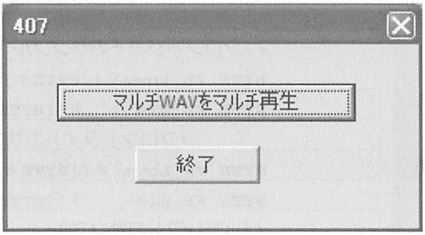
中級

サンプルプロジェクト　407　マルチ→マルチ

サンプルでは4つのチャンネルを収録したWAVファイルを読み込み、4チャンネルのDirectSoundバッファを作成、それを4つのスピーカー（前左右と後ろ左右）から出力します。

ポイント部分は当然、4チャンネルDirectSoundバッファの作成部分です。詳しくはTips414を参照してください。

▼実行結果



4チャンネルのDirectSoundバッファを作成できさえすれば、他の部分は通常の再生と同様です。

なお、WAVファイル側の作成に関しては、専用の波形編集ソフトを使用して作成しました。

408 マルチチャンネルWAVファイルをモノラルにダウンミックスする

●4チャンネル分のWAVデータを1チャンネルに詰め込む

上級

サンプルプロジェクト　408　マルチ→モノ　ダウンミックス

本サンプルでは4チャンネル分のWAVデータを1つにします。

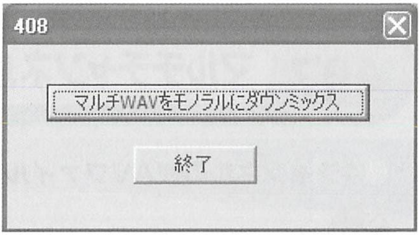
4チャンネルの特定のチャンネルのみを再生するのとは異なります（特定チャンネルのみの再生に関してはTips410を参照）。

このような処理はダウンミックスと言い、DirectSound関係というよりも、もっと一般的なWAVデータの操作が本質です。WAVデータの構造を理解していれば簡単にできることであり、4チャンネルに限らず5チャンネル7チャンル等でも同じことです。そして、ダウンミックスもモノラルにするのもステレオにするのも大した違いはありません。要するに、WAVデータの仕様を理解していればどのような組み合わせでも実装できるということです。逆に言うとWAVファイルが各チャンルのWAVデータをどのような配置で格納しているのかが分からないと、パターンごとに丸暗記しなくてはなりません（そしてそれはあまり意味がありません）。

WAVファイル内のWAVデータの配置等についてはTips417を参照してください。それを把握しているという前提で話を進めます。

ダウンミックスを行っているのは、118行から138行の部分です。本サンプルで使用しているWAVファイルは16ビットであることが分かっているので、次のようなコードを書きました。

▼実行結果



```
        for( DWORD i = 0; i < dwBufferSize/2; i++ )
       {
            //フロントレフトのサンプル
            BYTE FL_Low=＊((BYTE＊) (pWavData+i＊8+0) );
            BYTE FL_Hi=     ＊((BYTE＊) (pWavData+i＊8+1) );
                  //フロントライトのサンプル
            BYTE FR_Low= ＊((BYTE＊) (pWavData+i＊8+2) );
            BYTE FR_Hi=     ＊((BYTE＊) (pWavData+i＊8+3) );
            //リアレフトのサンプル
            BYTE RL_Low=＊((BYTE＊) (pWavData+i＊8+4) );
            BYTE RL_Hi=    ＊((BYTE＊) (pWavData+i＊8+5) );
            //リアライトのサンプル
            BYTE RR_Low= ＊((BYTE＊) (pWavData+i＊8+6) );
            BYTE RR_Hi=    ＊((BYTE＊) (pWavData+i＊8+7) );

            //16bitであることが分かっている。１サンプルに対し2バイト必要

            //4つのチャンネルをごちゃ混ぜにするのでモノラルだと聞き辛いかも
           ＊((BYTE＊)pBuffer+i＊2+0) =( FL_Low +  FR_Low + RL_Low + RR_Low );
           ＊((BYTE＊)pBuffer+i＊2+1) =( FL_Hi + FR_Hi + RL_Hi + RR_Hi );
        }
```

注意するのは、この形が全てではないということです。実際、異なるビット深度のWAVファイルではうまくいかないでしょう。

しかし、WAVデータの仕様が分かっていると簡単に修正することができます。今までのサンプルで、全てのWAVファイルに対応できていたのは、DirectSoundバッファとWAVファイルのフォーマットを同じものにしていたからです。WAVファイルのフォーマットをそのままバッファ側にコピーしていたので当然です。

そして、バッファへのWAVデータの詰め込みでも、バイト単位でコピーしていたので（バイト単位でコピーできたので）アライメントがずれることなど起こりようがなかっただけです。

もう一つ今まで上手くいっていた理由は、DirectSoundバッファ（のデータ領域）のサイズとWAVデータのサイズが一致していたので、バイト単位でそっくりそのまま単純にコピーすればいいだけだったことです。

しかし、本サンプルではその両方とも当てはまりません。DirectSoundバッファとWAVファイルのフォーマットも異なれば、データ部分のサイズも異なります。WAVファイルのフォーマットが4チャンルなのに、DirectSoundバッファは単純な1チャンネルで作成しています。当然、サイズもWAVファイルは4チャンル分あるのに、DirectSoundバッファ側は1チャンネル分しかありません。つまりWAVファイルの4分の1しかないのです。

そのため、波形データのコピーにはそれなりの工夫が必要になり、今までのように単純ではなく、しかも、WAVデータがどうようなビット深度なのかなどを気にする必要があり、それぞれの組み合わせでコードを微妙に異なるものにする必要も出てきます。

●コード解説

```
    for( DWORD i = 0; i < dwBufferSize/2; i++ )
    {
```

DirectSoundバッファ側のポインターを
＊((BYTE＊)pBuffer+i＊2+0)
＊((BYTE＊)pBuffer+i＊2+1)
のように計算しているので、ループの上限数をバッファサイズの半分にします。

そうしないとサイズの半分以上まで回ったときにアクセス違反が起こります。

```
        //フロントレフトのサンプル
        BYTE FL_Low=＊((BYTE＊) (pWavData+i＊8+0) );
        BYTE FL_Hi=    ＊((BYTE＊) (pWavData+i＊8+1) );
        //フロントライトのサンプル
        BYTE FR_Low= ＊((BYTE＊) (pWavData+i＊8+2) );
        BYTE FR_Hi=    ＊((BYTE＊) (pWavData+i＊8+3) );
        //リアレフトのサンプル
        BYTE RL_Low=＊((BYTE＊) (pWavData+i＊8+4) );
        BYTE RL_Hi=    ＊((BYTE＊) (pWavData+i＊8+5) );
        //リアライトのサンプル
        BYTE RR_Low= ＊((BYTE＊) (pWavData+i＊8+6) );
        BYTE RR_Hi=    ＊((BYTE＊) (pWavData+i＊8+7) );
```

この部分で、WAVデータ側の各バイトを取り出しています。16ビット4チャンネルのWAVデータはバイトにすると8バイトあります（Tips417参照のこと）。FL_Low、FL_Hiの2バイトで一つのチャンネルです。_Lowは16ビットのうち下位8ビット、_Hiは上位8ビットという意味です。あとの3チャンネルも同様です。

コードから分かると思いますが、4チャンネル・クアドロフォニックのデータ配置は、前左、前右、後左、後右の順序です（その他のタイプについてはTips417を参照）。

```
        //16bitであることが分かっている。1サンプルに対し2バイト必要

        //4つのチャンネルをごちゃ混ぜにするのでモノラルだと聞き辛いかも
       ＊((BYTE＊)pBuffer+i＊2+0) =( FL_Low +  FR_Low + RL_Low + RR_Low );
       ＊((BYTE＊)pBuffer+i＊2+1) =( FL_Hi +  FR_Hi + RL_Hi + RR_Hi );
    }
```

WAVデータから取り出した4チャンネルを1チャンル内に全て詰め込みます。2行あっても、それぞれが上位バイト、下位バイトなのでこれで1チャンネルです（16ビットなので）。

16ビットの場合は、4チャンルWAVデータ側は8バイト、DirectSoundバッファ側は2バイトとなります。

409 マルチチャンネルWAVファイルをステレオにダウンミックスする

●4チャンネル分のWAVデータを2チャンネルに詰め込む

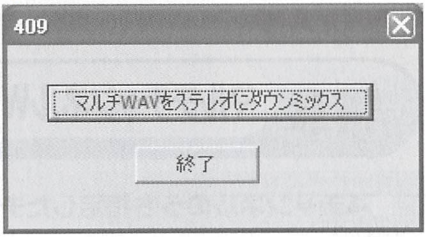
上級

サンプルプロジェクト　409　マルチ→ステレオ　ダウンミックス

本サンプルでは4チャンネル分のWAVデータを2チャンネルにします。このような処理はダウンミックスと言います。本質については直前のTipsを参照してください。また、直前のTipsで重要な事柄を述べているので必ず見るようにしてください。

やっていることの本質は、直前のTipsと全く同じです。WAVデータをDirectSoundバッファにコピーする際のバイトの取り出しと、計算が多少異なります。

▼実行結果



```
for( DWORD i = 0; i < dwBufferSize/2; i++ )
    {
        //フロントレフトのサンプル
        BYTE FL_Low=＊((BYTE＊) (pWavData+i＊8+0) );
        BYTE FL_Hi=     ＊((BYTE＊) (pWavData+i＊8+1) );
            //フロントライトのサンプル
        BYTE FR_Low= ＊((BYTE＊) (pWavData+i＊8+2) );
        BYTE FR_Hi=     ＊((BYTE＊) (pWavData+i＊8+3) );
        //リアレフトのサンプル
        BYTE RL_Low=＊((BYTE＊) (pWavData+i＊8+4) );
        BYTE RL_Hi=     ＊((BYTE＊) (pWavData+i＊8+5) );
        //リアライトのサンプル
        BYTE RR_Low= ＊((BYTE＊) (pWavData+i＊8+6) );
        BYTE RR_Hi=     ＊((BYTE＊) (pWavData+i＊8+7) );
```

ここまでは直前のTipsと同様です。

```
        //フロント2つで1チャンネル　リア2つで1チャンネル　という具合に分けたが、分け方は別に
もあるだろう
       ＊((BYTE＊)pBuffer+i＊4+0) =( FL_Low +  FR_Low  );
       ＊((BYTE＊)pBuffer+i＊4+1) =( FL_Hi +  FR_Hi  );

       ＊((BYTE＊)pBuffer+i＊4+2) =( RL_Low +  RR_Low  );
```

```
        *((BYTE*)pBuffer+i*4+3) =( RL_Hi +  RR_Hi  );
    }
```

DirectSoundバッファー側が今度はステレオなので、2チャンネルあります。したがってDirectSoundバッファー側には4バイトのデータまで格納できます。

4チャンネルをどのようなパターンで2チャンネルに詰め込むかは、いろいろあると思いますが、ここではとりあえず、「DirectSoundバッファー側の左のチャンネルには、WAVデータの前左と後左のチャンネルを」「DirectSoundバッファ側の右のチャンネルには、WAVデータの前右と後右のチャンネル」という風にしました。

410 マルチチャンネルWAVファイルの特定チャンネルを再生する

●4チャンネルのうち指定したチャンネルのみ再生する

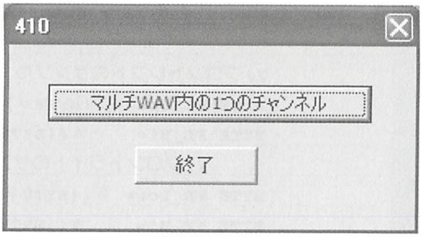
上級

サンプルプロジェクト　410　マルチ→1チャンネル

4チャンネルのWAVデータのうち、前左のチャンネルのみを再生します。

まずTips408を参照してください。DirectSoundバッファへWAVデータの各バイトをコピーする部分で、一個のチャンネル分しかコピーしないという簡単な変更です。

▼実行結果



```
*((BYTE*)pBuffer+i*2+0) =FL_Low;
*((BYTE*)pBuffer+i*2+1) =FL_Hi;
```

前左チャンネルではなく、例えば後右チャンネルの音を出したいのであれば、

FL_Low、FL_Hi

を

RR_Low、RR_Hi

にすればいいだけです。これは問題ないでしょう。

411　4つのモノラルWAVファイルをマルチチャンネル再生する

モノラルWAVファイル4から一つのマルチバッファを作成する

上級

サンプルプロジェクト　411　モノx4→マルチ

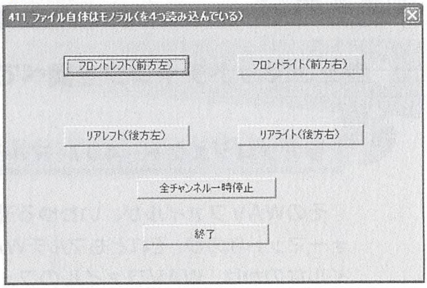
このサンプルは、4つのWAVファイルを読み込んで、それらを1チャンネルのDirectSoundバッファ4つに詰め込みます。

DirectSoundバッファバッファは、あくまで1チャンネルであるものを4つ用意すれことに注意してください。Tips407のようにバッファ自体が4チャンネル用であるのとは異なります。

ポイント部分は当然、特定のチャンネル用（例えば左前、右後のチャンネル）のDirectSoundバッファ作成部分です。詳しくはTips415を参照してください。

特定のチャンネル用DirectSoundバッファを作成できさえすれば、つまり意図したスピーカーだけから出力するようなDirectSoundバッファさえ作成できれば、他の部分は通常の再生と同様です。

▼実行結果



412　6つのモノラルWAVファイルを5.1チャンネルで鳴らす

モノラルWAVファイル6から一つのマルチバッファを作成する

上級

サンプルプロジェクト　412　5.1チャンネル

このサンプルは、6つのWAVファイルを読み込んで、それらを1チャンネルのDirectSoundバッファ6つに詰め込みます。

DirectSoundバッファは、あくまで1チャンネルであるものを6つ用意すれことに注意してください。

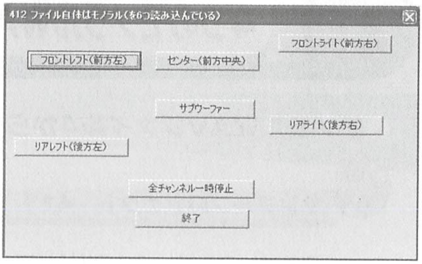
Tips407のようにバッファ自体が4チャンネル用であるのとは異なります。

ポイント部分は当然、特定のチャンネル用（例えば左前、右後のチャンネル）のDirectSoundバッファ作成部分です。詳しくはTips415を参照してください。

特定のチャンネル用DirectSoundバッファを作成できさえすれば、つまり意図したスピーカーだけから出力するようなDirectSoundバッファさえ作成できれば、

他の部分は通常の再生と同様です。

▼実行結果



412 ファイル自体はモノラル（を6つ読み込んでいる）
フロントライト（前方右）
フロントレフト（前方左）
センター（前方中央）
サブウーファー
リアライト（後方右）
リアレフト（後方左）
全チャンネル一時停止
終了

413 WAVファイルがマルチチャンネルかどうか調べる

●フォーマットチャンクを調べてマルチチャンネルの判断をする

中級

参考プロジェクト　407 マルチ→マルチ

そのWAVファイルが、いわゆる通常のフォーマットなのか、それともマルチWAVファイルなのかは、WAVファイルのフォーマットを読み込み、その.Formatメンバのさらにw FormatTagメンバを調べれば分かります。

サンプルでは次のように判断しています。

```
//フォーマットを読み込む
if( mmioRead( hMmio, (CHAR*) &pcmWFEXBL,
          sizeof(WAVEFORMATEXTENSIBLE)) != sizeof(WAVEFORMATEXTENSIBLE) )
{
    MessageBox(0,"WAVフォーマットの読み込み失敗","",MB_OK);
}
//3チャンネル、クアドロフォニック(4チャンネル)、5.1チャンンル等のマルチチャンネルフォーマッ
トであることの確認
if( pcmWFEXBL.Format.wFormatTag != WAVE_FORMAT_EXTENSIBLE)
{
    MessageBox(0,"本サンプルはリニアPCM(非圧縮)フォーマット、かつ、マルチチャンネルWAVを
想定しています                ","",MB_OK);
    return E_FAIL;
}
```

414 マルチチャンネルのセカンダリバッファを作成する

●WAVE_FORMAT_EXTENSIBLE構造体を使う

中級

参考プロジェクト　407 マルチ→マルチ

マルチチャンネルと言っても特別なことはなく、コードは通常のバッファ（モノラルかステレオ）と同じです。

```
DSBUFFERDESC dsbd;
ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
dsbd.dwSize = sizeof(DSBUFFERDESC);
dsbd.dwFlags = DSBCAPS_CTRLVOLUME | DSBCAPS_CTRLFREQUENCY;
dsbd.dwBufferBytes = dwWavSize;
dsbd.lpwfxFormat = (WAVEFORMATEX＊)&pcmWFEXBL.Format;

g_pDSound->CreateSoundBuffer( &dsbd, &g_pDSBuffer[ dwSoundIndex ], NULL );
```

dwWavSizeはWAVデータのサイズです。pcmWFEXBLは、WAVファイルのフォーマットをそのまま読み込んだだけの構造体ですが、型はWAVEFORMATEXTENSIBLE型です（通常のWAVEFORMATEX型とは違います）。簡単です。

ただ、これが簡単であるのは、WAVファイル側がマルチチャンネルである場合です。WAVファイル側がマルチチャンネルになっている場合は、フォーマット、データ共に単にバッファにコピーすればいいだけなのでそうなります。

そうではなく、WAVファイル自体は通常のもので、DirectSoundバッファだけをマルチにしたい場合は、少々面倒になります。それについては次のTipsで解説します。

415 特定チャンネルのセカンダリーバッファを作成する

●複数のスピーカーのうち、意図したスピーカー用のバッファを作成する

上級

参考プロジェクト　411 モノx4→マルチ

特定のチャンネル用のDirectSoundバッファ、つまり、特定のスピーカーからのみ音を出力するようなDirectSoundバッファを作成するには、前段として、WAVE

FORMATEXTENSIBLE構造体を自前に初期化しなくてはなりません。

マルチWAVファイルからマルチバッファを作成するときは、ファイル内のフォーマットをコピーするだけだったので、この作業が不要でした。

例えばモノラルWAVファイルであるい場合、それと同じ方法でのフォーマットをそのままコピーしてしまうとDirectSoundバッファがモノラルフォーマットになってしまうので、その方法がとれないのは明らかでしょう。

特定のチャンネル用のDirectSoundバッファのフォーマット自体はマルチフォーマットである必要があります。

コードを示すとしたら、LoadSound関数全体ということになってしまうので、参考程度にコードを掲載します。

```
//チャンル数などの追加情報を含む"EXTENSIBLE"フォーマットを作成する
        memcpy( &pcmWFEXBL.Format, &pcmWFEX, sizeof(pcmWFEX) );
        pcmWFEXBL.Format.wFormatTag=WAVE_FORMAT_EXTENSIBLE;
        pcmWFEXBL.dwChannelMask=dwChannelMask;
        pcmWFEXBL.Format.cbSize=sizeof(WAVEFORMATEXTENSIBLE) -
sizeof(WAVEFORMATEX);
        pcmWFEXBL.Samples.wSamplesPerBlock=pcmWFEX.wBitsPerSample;
        pcmWFEXBL.Samples.wValidBitsPerSample=pcmWFEX.wBitsPerSample;

memcpy(&pcmWFEXBL.SubFormat,&KSDATAFORMAT_SUBTYPE_PCM,sizeof(pcmWFEXBL.SubFo
rmat));
         if( MMSYSERR_NOERROR != mmioAscend( hMmio, &ckInfo, 0 ) )
        {
            return E_FAIL;
        }
        // WAVファイル内の音データ(ヘッダー以外のデータ、波形データ)の読み込み
        ckInfo.ckid = mmioFOURCC('d', 'a', 't', 'a');
         if( MMSYSERR_NOERROR != mmioDescend( hMmio, &ckInfo, &riffckInfo,
MMIO_FINDCHUNK ) )
        {
                MessageBox(0,"dataチャンクが見つかりません","",MB_OK);
                return E_FAIL;
        }
        dwWavSize = ckInfo.cksize;
      // そのWAVフォーマットを基に、新たに作成した"EXTENSIBLE"フォーマットのDirectSoundセ
カンダリーバッファ作成
        DSBUFFERDESC dsbd;
      ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
      dsbd.dwSize = sizeof(DSBUFFERDESC);
      dsbd.dwFlags = DSBCAPS_CTRLVOLUME | DSBCAPS_CTRLFREQUENCY;
      dsbd.dwBufferBytes = dwWavSize;
      dsbd.guid3DAlgorithm = DS3DALG_DEFAULT;
      dsbd.lpwfxFormat = (WAVEFORMATEX*)&pcmWFEXBL;
```

```
LPDIRECTSOUNDBUFFER ptmpBuffer;
if( FAILED( g_pDSound->CreateSoundBuffer( &dsbd, &ptmpBuffer, NULL ) ) )
{
   MessageBox(0,"セカンダリバッファ作成失敗","",MB_OK);
   return E_FAIL;
}
```

416 WAVE_FORMAT_EXTENSIBLE(およびWAVEFORMATEX)構造体

WAVE_FORMAT_EXTENSIBLE構造体を理解する

中級

WAVE_FORMAT_EXTENSIBLE構造体が持つメンバと、その意味は次のとおりです。

なお、ぱっと見るとメンバの数が一見少なそうに見えますが、Formatメンバは WAVEFORMATEX型であるので、入れ子状にまだまだメンバが存在します。

```
typedef struct {
  WAVEFORMATEX  Format;
  union {
    WORD  wValidBitsPerSample;
    WORD  wSamplesPerBlock;
    WORD  wReserved;
  } Samples;
  DWORD    dwChannelMask;
  GUID     SubFormat;
} WAVEFORMATEXTENSIBLE, *PWAVEFORMATEXTENSIBLE;
```

Format

WAVEFORMATEX型のフォーマット構造体です（後で解説）。

Samples

共用体なのでwValidBitsPerSampleかwSamplesPerBlockのどちらかになります。意味はそれぞれ、サンプルのビット深度、ブロックのサンプル数です。通常はwValidBitsPerSampleのほうに意味を持たせるので、大抵はビット深度を意味します。したがってリニアPCMであれば8または16の値を取ります。

dwChannelMask

チャンネルの形態です。言い換えると意図されたスピーカー配置とも言えます。3スピーカー、クアドロフォニック、5.1チャンネル、7.1チャンネルなど、様々な形態が想定され、それらの定数（DWORD値）が用意されています。値と意味は次のとおりです。

値	意味
SPEAKER_FRONT_LEFT 0x1	前左スピーカー
SPEAKER_FRONT_RIGHT 0x2	前右スピーカー
SPEAKER_FRONT_CENTER 0x4	前中央スピーカー
SPEAKER_LOW_FREQUENCY 0x8	サブウーファースピーカー（スーパーウーファーとも言う）
SPEAKER_BACK_LEFT 0x10	後左スピーカー
SPEAKER_BACK_RIGHT 0x20	後右スピーカー
SPEAKER_FRONT_LEFT_OF_CENTER 0x40	前中央やや左スピーカー
SPEAKER_FRONT_RIGHT_OF_CENTER 0x80	前中央やや右スピーカー
SPEAKER_BACK_CENTER 0x100	後中央スピーカー
SPEAKER_SIDE_LEFT 0x200	横左スピーカー
SPEAKER_SIDE_RIGHT 0x400	横右スピーカー
SPEAKER_TOP_CENTER 0x800	天井中央スピーカー
SPEAKER_TOP_FRONT_LEFT 0x1000	天井前左スピーカー
SPEAKER_TOP_FRONT_CENTER 0x2000	天井前中央スピーカー
SPEAKER_TOP_FRONT_RIGHT 0x4000	天井前右スピーカー
SPEAKER_TOP_BACK_LEFT 0x8000	天井後左スピーカー
SPEAKER_TOP_BACK_CENTER 0x10000	天井後中央スピーカー
SPEAKER_TOP_BACK_RIGHT 0x20000	天井後右スピーカー

SubFormat

WAVファイルの中にはサブフォーマットを持つものがありますが、本書および通常では扱いません。

では次に、Formatメンバの型でもあるWAVE_FORMAT_EXTENSIBLE構造体のメンバとその意味を解説します。

```
typedef struct {
    WORD  wFormatTag;
    WORD  nChannels;
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD  nBlockAlign;
    WORD  wBitsPerSample;
    WORD  cbSize;
} WAVEFORMATEX;
Format.wFormatTag
```

WAVファイル内、フォーマットチャンクのタグです。本書ではこれを見ることにより、そのWAVファイルが通常のWAVなのかマルチWAVなのかということの判断に使用しています。

タグの種類ですが、困ったことに130以上の種類があります。しかし実際にお目にかかるのは、通常のWAVファイルであれば、タグはWAVE_FORMAT_PCM、マルチWAVであれば、タグはWAVE_FORMAT_EXTENSIBLEくらいです（本書ではこの2種類だけです）。

Format. nChannels

チャンネル数です。モノラルなら1、ステレオなら2、クアドロフォニックなら4、5.1chなら6、7.1chなら8…などとなります。

Format. nSamplesPerSec

1秒あたりのサンプル個数、いわゆるサンプリング周波数です。22050（hz）、44100（hz）などの値を取ります。

Format. nAvgBytesPerSec

1つのチャンネルについて、1秒間のバイト数です。様々なパターンがありますが、例えば

8bit　22khzなら　8×22050=176400
8bit　44.1khzなら　8×44100=352800
16bit　22khzなら　16×22050=352800
16bit　44.1khzなら　16×22050=705600

などとなります。もちろん組み合わせはこれだけではありません。

なお、この値は1つのチャンネル分のみを表しています。おそらくそれがAvg(Average)の意味でしょう。したがって、全体の1秒間のバイト数はこれにチャンネル数を乗算した値になります。

Format. nBlockAlign

ブロックのアライメントです。単位はバイトです。アライメントと言われても分かり辛いでしょう。

要するに、チャンネル数とビット深度を加味したときのサンプル1個の論理サイズです（これのほうが分からないかな？）。

式で表現すれば確実ですね。

チャンネル数（nChannels）×ビット深度（wBitsPerSample）/8を計算した値です。

したがって、例えばモノラル8ビットなら1×1=1、モノラル16ビットなら1×2=2、ステレオ16ビットなら、2×2=4となり、4チャンネル16ビットなら4×2=8となります。

Format. wBitsPerSample

1つのサンプルに何ビット使用しているかという意味です。いわゆるビット深度です。8（ビット）、16（ビット）などとなります。

Format. cbSize

これは拡張部分のメンバのバイト数です。通常であれば0ですが、ここでのようにWAVE_FORMAT_EXTENSIBLE構造体の中にある場合は22となります（23以上になるとしたら圧縮PCM等であり、本書の対象外です）。

417 WAVデータフォーマットの詳細

●WAVE_FORMAT_EXTENSIBLE構造体を理解する　中級

ここでは、WAVデータがどのような構成になっているか、サンプル（波形データ）をどのように羅列しているか、また、2チャンネル以上ある場合のチャンネルごとのサンプルの順序はどうなっているかなどを解説します。

最も単純なのはやはりモノラルで、しかも

8bitのモノラルです。8bitモノラルはフレームとバイトが1対1に対応します。8bitステレオになるとチャンネルが2つになるので、その分、1フレームに対し2バイト必要になります。

さらに、8bitクアドロ（4チャンネル）になると、1フレームに対し、4バイト必要になり、8bit　5.1chにいたっては、6つのチャンネルがあるので、1フレームに対し、6バイト必要になります。

▼【モノラル　8bit】

フレーム	0	1	2	3	4	5	6	7	8	9	10	11
バイト	0	1	2	3	4	5	6	7	8	9	10	11

▼【ステレオ　8bit】

フレーム	0		1		2		3		4		5	
チャンネル	左	右	左	右	左	右	左	右	左	右	左	右
バイト	0	1	2	3	4	5	6	7	8	9	10	11

▼【クアドロ　8bit】

フレーム	0				1				2			
チャンネル	前左	前右	後左	後右	前左	前右	後左	後右	前左	前右	後左	後右
バイト	0	1	2	3	4	5	6	7	8	9	10	11

▼【5.1チャンネル　8bit】

フレーム	0						1					
チャンネル	前左	前右	中央	ウーハ	後左	後右	前左	前右	中央	ウーハ	後左	後右
バイト	0	1	2	3	4	5	6	7	8	9	10	11

さらに、16ビットになるとそれぞれのフレームが単純に2倍のバイト数になります。一つのチャンネルは、2バイトで表現されます。最初のバイトは下位バイト、後のバイトは上位バイトになります（リトルエンディアン）。

▼【モノラル　16bit】

フレーム	0		1		2		3		4		5	
上位、下位	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi
バイト	0	1	2	3	4	5	6	7	8	9	10	11

▼【ステレオ　16bit】

フレーム	0				1				2				3			
チャンネル	左		右		左		右		左		右		左		右	
上位、下位	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi
バイト	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

▼【クアドロ　16bit】

フレーム	0								1							
チャンネル	前左		前右		後左		後右		前左		前右		後左		後右	
上位、下位	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi
バイト	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

▼【5.1チャンネル　16bit】

フレーム	0												1											
チャンネル	前左		前右		前中央		ウーハ		後左		後右		前左		前右		前中央		ウーハ		後左		後右	
上位、下位	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi	Low	Hi
バイト	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

16bit5.1chでは、1フレームに必要なバイト数はなんと12バイトです。これでサンプリング周波数が４４１００hzなら１２×４４１００ですから、一秒間の再生には529200バイトも必要になります。

One Point

なお、フレーム＝チャンネルxサンプル　です。したがって、モノラルの場合のみ　1フレーム＝1サンプル、そして特に、モノラル8bitの場合のみ　1フレーム＝1バイト　になります。

第 8 章
418～430
録音の極意

8-1　サウンドのキャプチャー（418～430）

418 サウンドのキャプチャーとは？

●サウンドキャプチャーとは「録音」のこと

初級

サウンドをキャプチャー（捕らえる）するとは「録音」のことです。録音と言っても何をどう録音するのでしょう？

録音する音はWAV音で、マイクからの声もあるでしょうし、メディアプレイヤーで再生中のWAV音もそうです。そして、それらを通常はファイルにするのが普通です。

録音するソースは、wavファイル、同じwavで圧縮ADPCM、mp3、wma...etc様々なファイル形式でもキャプチャーできます。ただし、midファイルはキャプチャーできません。

One Point

特に学習中は市販のCD等の楽曲を何らかのプレイヤーで再生し、それをキャプチャーすることという作業が多いと考えれられます。キャプチャーしてファイルに落とすのは自由ですが（これすら禁止している場合もあります）、そのファイルを著作権と使用権の範囲内で使用することは言うまでもありません（再配布したりすると普通は著作権・使用権に抵触します）。

もちろん、録音するソース側も自分で制作したサウンドである場合は、なんら気にすることはありません。

419 今スピーカーから出ている音を録音する

●“スピーカーから出る音“について考える

初級

“スピーカーから音が出る”ということは、PC内でなんらかのデータをレンダリングしているわけですから（midiデータと言えども、最終的にはWAVベースの音色にマッピングされて出力されるわけですから）、必ず“捕まえる”ことが出来そうな気がしますね。

しかし、（DirectSoundでは）そうではありません。

メッセージベースのサウンド（元のデータ自体は波形ではなく単なるインデックスのようなもの）がWAVにマッピングされてから“捕まえれば”いいわけですが、残念ながらその段階の音まで捕まえてくれるほどDirectSoundの手は長くはありません。

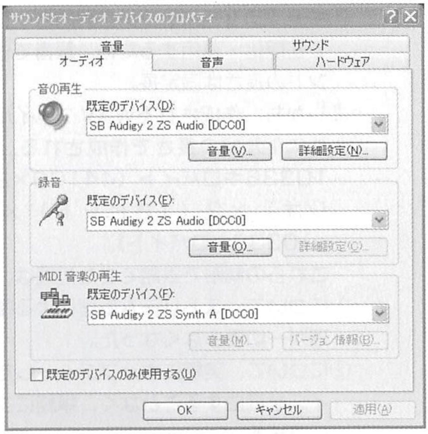
したがって、理論的には可能ですが、DirectSoundの仕様的には不可能ということになります。

あと、サウンドファイルではなく“マイク”からのサウンドもキャプチャーの入力になり得ます。マイクからの音声をキャプチャーしたい場合は、PCのデバイス設定が次のようになっていることを確認してください。

まずコントロールパネル「サウンドとオーディオデバイス」を開きます。「サウンドとオーディオデバイスのプロパティ」ダイアログが出るので、その「オーディオ」タブをクリックします。

そして、「録音」の「音量」をクリックし、「レコーディングコントロール」ダイアログを出します。

▼実行結果


サウンドとオーディオ デバイスのプロパティ
?
X
音量
サウンド
オーディオ
音声
ハードウェア
音の再生
既定のデバイス(D):
音量(V)...
詳細設定(N)...
録音
既定のデバイス(E):
音量(O)...
詳細設定(O)...
MIDI 音楽の再生
既定のデバイス(F):
音量(M)...
バージョン情報(B)...
OK
キャンセル
適用(A)

音の再生　既定のデバイス(D): SB Audigy 2 ZS Audio [DCC0]

録音　既定のデバイス(E): SB Audigy 2 ZS Audio [DCC0]

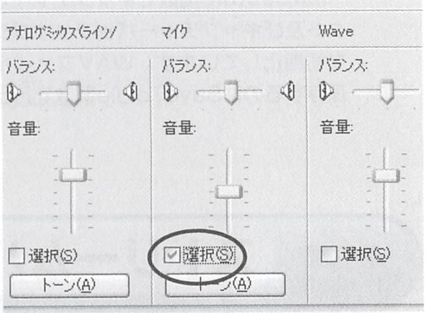
MIDI 音楽の再生　既定のデバイス(F): SB Audigy 2 ZS Synth A [DCC0]

☐既定のデバイスのみ使用する(U)

“マイク”にチェックがはいっていないと、マイクの入力をキャプチャーできませんので、チェックしてください。もちろん、メディアプレイヤー等の音をキャプチャーしたい場合は、“マイク”のチェックを外し、“Wave”をチェックすることになります。

実はDirectSoundではなく、Windows Multimedia MIDIサービスであればmidiのキャプチャが可能です。しかし、プラットフォームSDKは本書の対象外なので踏み込みません。

▼実行結果


アナログミックス(ライン)
マイク
Wave
バランス:
音量:
トーン(A)

アナログミックス(ライン)　☐選択(S)

マイク　☑選択(S)

Wave　☐選択(S)

420 静的バッファで録音（簡単版）

録音してWAVファイルを作成する

上級

サンプルプロジェクト　420　録音（単純版）

本サンプルは、メディアプレイヤー等から鳴っているWAVサウンドやマイクからの音声をキャプチャーするものです。

Tips419で述べたとおり、WAV入力とマイク入力はそれぞれOS側で切り替える必要があります。

このサンプルは、とにかくすみやかな理解ができるよう極力単純化しています。そのため

次の制限があります。

①キャプチャーバッファを静的バッファとして初期化・運用するため、録音できるサウンドの長さは固定長。

②しかも、作成されるWAVファイルは無条件に1分間の長さで作成される。常に約10336キロバイト（44100×2（ステレオ）×2（16ビット）×60秒=10336キロバイト)。

これらの制限で省略できた処理は、

①について、ストリーム処理による複雑な処理が一切要らなくなった。

②について、実際にキャプチャーした音のサイズを気にすることなく、単純にファイル化できる。

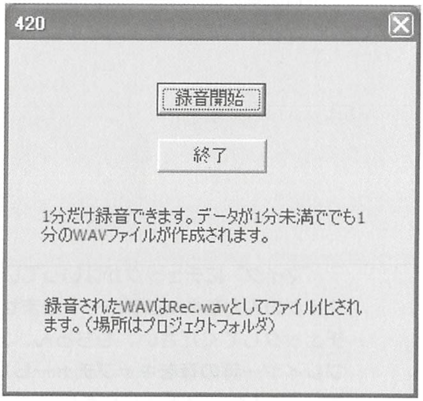
という点です。

InitCapture関数でキャプチャーオブジェクト及びキャプチャーバッファオブジェクトを初期化しています。WAVファイルとして保存するのはSaveToFile関数で行っています。ポイントはこの2つの関数だけです。

それぞれの具体的なコーディングについては他のTipsを参照してください。

また、ストリームを行い、ファイルも実際のキャプチャ分のみ作成するサンプルは次のTips421にあります。

▼実行結果



1分だけ録音できます。データが1分未満ででも1分のWAVファイルが作成されます。

録音されたWAVはRec.wavとしてファイル化されます。（場所はプロジェクトフォルダ）

421 ストリームバッファで録音（完全版）

●ストリーム録音してWAVファイルを作成する

上級

サンプルプロジェクト　421　録音（完全版）

本サンプルは、直前のサンプルの制限を解消した改良版です。したがって、録音できる時間は無制限で、また、吐き出すWAファイルは実際にキャプチャーしたぶんだけの長さになります。ストリーミングの原理、利点については7-2章を参照してください。ストリーミングの仕組みは再生編と全く同じです。

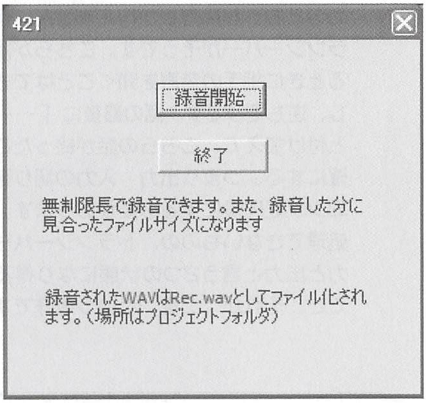
ただ、再生編では通知領域を4分割していましたが、ここでは2分割にしました。通知区間をキャプチャーカーソルが通過したときにイベントメッセージを発行するように設定し、その通知を受け取った瞬間に、今ちょうど通過した区間から読み込むことにより、これ以上ない安全なタイミングで読み込みが出来ます。読み込みカーソルなど全く不用です。これについても再生編で解説しています（再生編では読み込みではなく、書き込みですが）。

InitCapture関数でキャプチャーオブジェクト及びキャプチャーバッファオブジェクトを初期化しています。イベントメッセージを

ハンドルしているのはNotifyProc関数です。WAVファイルとして保存するのはSaveToFile関数で行っています。ポイントはこの3つの関数だけです。

なお、ストリーミングの場合は、再生中逐次ファイルにアクセスするのでSaveToFile関数はイベントメッセージを受け取った数だけ何度も呼ばれます。そのため、プログラムの開始直後にファイルを作成しRIFFヘッダー等波形データ以外のチャンクを埋め、そしてファイルをクローズすることなくファイルポインタを再生中に使用しています。ファイルをクローズするのはプログラム終了時です。

▼実行結果



無制限長で録音できます。また、録音した分に見合ったファイルサイズになります

録音されたWAVはRec.wavとしてファイル化されます。(場所はプロジェクトフォルダ)

422 キャプチャーデバイスってサウンドカードと違うの？

キャプチャーデバイスの意味を把握する

初級

キャプチャーデバイスとサウンドカード、通常は同じです。サウンドカードを録音時に使用するときの呼び方ですので再生デバイスであり同時にキャプチャーデバイスです。

ただし、“通常は”と言葉を濁したのは、異なる場合もあるからです。ヘッドセットマイクなどはキャプチャーデバイスであって再生デバイスではありません。

つまり、たまたま普通の環境では同じ機器（サウンドカード）を指すだけのことであり、概念的には異なります。コントロールパネルで再生デバイスと録音デバイスを別々のものにセットできることからも、2つの概念がある必要があります。

423 全二重、半二重とは？

全二重、半二重の意味を理解する

初級

全二重とは、出力を行いながら同時に入力も行えるという意味です。良い例としては電話がそうです。電話での通話は、こちらが話しているときも相手の声が聞えるように、出力と入力が同時に行えます。

一方、半二重とは出力と入力が同時に行え

ないという意味です。ある瞬間には出力か入力のどちらか片方しか行えません。例えばトランシーバーがそうです。こちらが話しているときに相手の音声を聞くことはできませんし、逆もそうです。話の最後に「……どうぞ」と付け加えて、こちらの話が終ったことを明確にする、つまり出力・入力の切り替えタイミングを円滑にする必要があります。同時に処理できないものの、トランシーバーでも入力と出力と言う2つの状態になり得ることはでき、それが“二重”という意味です。

DirectSoundでは全二重をサポートしています。ただし、サウンドカード側がサポートしている場合に限ります。

全二重のインターフェイスは、IDirectSoundFullDuplex8です。FullDuplexは全二重という意味です。ちなみに半二重を英語で言うとHalfDuplexです。全二重を実装すると、たとえばマイクからの音声を録音しながら、同時にスピーカーからもその音声を鳴らすことができます。

なお、本書では全二重の実装は行いません。

424 キャプチャーデバイスを作成する

●DirectSoundCaptureCreate8ヘルパー関数を使用する　初級

参考サンプルプロジェクト　420　録音（単純版）　421　録音（完全版）

キャプチャーデバイスの作成は、DirectSoundCaptureCreate8関数により作成します。インターフェイス名が付かないことから分かるように、DirectSoundCaptureCreate8関数はCOMインターフェイスのメソッドではなく、グローバルな単体の関数です。このようにCOMインターフェイスの外にあり、COMオブジェクトを作成などを行う関数はヘルパー関数と呼ばれます。

サンプルでのコードは次のとおりです。

●宣言部分

```
LPDIRECTSOUNDCAPTURE8 g_DSCapture=NULL;//DirectSoundキャプチャーオブジェクト
```

●作成部分

```
if(FAILED(DirectSoundCaptureCreate8(&DSDEVID_DefaultCapture,&g_DSCapture,NUL
L)))
{
    MessageBox(0,"キャプチャオブジェクトの作成に失敗しました","",MB_OK);
    return E_FAIL;
}
```

ヘルパー関数を使用しないCOMオブジェクトの作成方法は、拙書『ゲームコーディングvol.1』（工学社刊）で解説しています。

425 キャプチャバッファを作成する

●IDirectSoundCaptureBufferからIDirectSoundCaptureBuffer8をQueryInterfaceする

初級

参考サンプルプロジェクト　420　録音（単純版）　421　録音（完全版）

作成の直接のコードは簡単ですが、フォーマット構造体とDSCBUFFERDESC構造体を作成する部分が面倒です。

サンプルでは次のように作成しています。

●宣言部分

```
LPDIRECTSOUNDCAPTURE8 g_DSCapture=NULL;//DirectSoundキャプチャーオブジェクト
LPDIRECTSOUNDCAPTUREBUFFER8 g_DSCaptureBuffer=NULL;//DirectSoundキャプチャーバッファオブジェクト
WAVEFORMATEX g_WaveFormat={0};//このWAVフォーマットでキャプチャする
```

●作成部分

```
    //録音するフォーマットは44.1khz 16bit ステレオ
    g_WaveFormat.wFormatTag=WAVE_FORMAT_PCM;
    g_WaveFormat.nChannels=2;
    g_WaveFormat.wBitsPerSample=16;
    g_WaveFormat.nAvgBytesPerSec=176400;
    g_WaveFormat.nSamplesPerSec=44100;
    g_WaveFormat.nBlockAlign=4;//2ch * (16bit/8);
    g_WaveFormat.cbSize=0;

    //キャプチャバッファ作成
    DSCBUFFERDESC desc={0};

    desc.dwSize = sizeof(DSCBUFFERDESC);
    desc.dwFlags = 0;
    desc.dwBufferBytes = g_WaveFormat.nAvgBytesPerSec * 60;//60秒ぶんだけ確保
    desc.dwReserved = 0;
    desc.lpwfxFormat = &g_WaveFormat;
    desc.dwFXCount = 0;
    desc.lpDSCFXDesc = NULL;

    LPDIRECTSOUNDCAPTUREBUFFER ptmp=NULL;
    g_DSCapture->CreateCaptureBuffer(&desc,&ptmp,NULL);
    if(NULL == ptmp)
    {
        MessageBox(0,"キャプチャバッファ作成失敗","",MB_OK);
        return E_FAIL;
    }
    ptmp->QueryInterface(IID_IDirectSoundCaptureBuffer8,(VOID*
*)&g_DSCaptureBuffer);
    if(NULL == g_DSCaptureBuffer)
    {
        MessageBox(0,"キャプチャバッファ8取得失敗","",MB_OK);
        return E_FAIL;
    }
```

g_WaveFormatはWAVEFORMATEX型の構造体です。これに録音フォーマットを格納します。

DSCBUFFERDESC型の記述用構造体descを用意し、これに作成したフォーマットg_WaveFormatを入れ、その他キャプチャーバッファ固有のパラメーターもセットします。

WAVEFORMATEX型の各メンバの意味は、Tips416で詳細に解説していますので理解していない読者はそちらを参照してください。

バッファの作成が2段階になっているのが分かるでしょうか？

PDIRECTSOUNDCAPTUREBUFFER型のインターフェイスポインターをまず取得します。一時的なインターフェイスポインターで構わないのでローカルに宣言しています。

```
LPDIRECTSOUNDCAPTUREBUFFER ptmp=NULL;
```

そのptmpにQueryInterfaceしてようやくLPDIRECTSOUNDCAPTUREBUFFER8型（8がつく）のバッファインターフェイスを取得しています。

```
ptmp->QueryInterface(IID_IDirectSoundCaptureBuffer8,(VOID*
*)&g_DSCaptureBuffer);
```

このようにするのは、DirectX7以前の古いDirectSoundオブジェクトとの互換性のためです。仕様だと思って納得しましょう。

426 ソース側とバッファ側のフォーマットは同じである必要はない

●キャプチャバッファのーフォーマットを独立に設定できることの意味合いを理解する　初級

入力ソース側、すなわち再生中のWAVあるいはマイクからの音声のフォーマットはなんでもよく、キャプチャーバッファーのフォーマットと同一である必要はありません。

キャプチャーバッファーのフォーマットは「キャプチャーしたWAVデータがそのフォーマットになる」という意味であって、例えば入力側が22050hz8bitでもキャプチャーバッファを4410016bitで作成しているのであればキャプチャー後のWAVは4410016bitになります。

もちろん音質は入力ソース以上にはなりませんので、4410016bit　用のメモリー空間があるのに格納されるデータは22050hz8bit分だけであり、この例の場合メモリの3/4は無駄ということになります。

逆に、ソース側のフォーマットが高解像度でバッファ側が低解像度の場合、これはこれで今度はカーネルミキサーによるダウンミックスが行われてしまうので、データ的に言えば入力ソースより音質が低下してしまうことになります（ただし体感的には個人差により気にならない場合もあるでしょう）。

そのようなメモリの無駄や（数字上の）音質低下が気になる場合に限り、同じフォーマットにこだわることになるでしょうか。

427 キャプチャーを開始する

●IDirectSoundCaptureBuffer8::Startメソッドでキャプチャーを開始する　初級

参考サンプルプロジェクト　420　録音（単純版）　421　録音（完全版）

キャプチャーを開始するには、IDirectSoundCaptureBuffer8::Startメソッドにより行います。当然、キャプチャーバッファが適正に初期化されている必要があります。

Tips420でのコードは次のようになります。

```
case IDC_BUTTON1:
    g_DSCaptureBuffer->Start(NULL);
return TRUE;
```

一方、Tips421では次のようにしています。

```
case IDC_BUTTON1:
        g_DSCaptureBuffer->Start(DSCBSTART_LOOPING);
return TRUE;
```

Tips420ではNULLを指定し、Tips421ではDSCBSTART_LOOPINGを指定しています。DSCBSTART_LOOPINGはキャプチャーをループさせるとう意味で、NULLはループせずに一回限りでいいという意味です。Tips421は、キャプチャーをストリーミングで運用するので、つまり小さいバッファに繰り返しキャプチャーする必要があるので、ループさせる必要があるのです。

それに対しTips420はストリーミングではなく運用としては静的バッファですので、循環させる必要がありません。キャプチャーはバッファが一杯になり次第終了させ、一旦バッファに格納されたデータが上書きされることもありません。

428 キャプチャーを停止する

●IDirectSoundCaptureBuffer8::Stopメソッドでキャプチャーを停止する　初級

参考サンプルプロジェクト　420　録音（単純版）　421　録音（完全版）

キャプチャーの停止は、IDirectSoundCaptureBuffer8::Stopメソッドで行います。

サンプルでのコードは次のようになります。

```
g_DSCaptureBuffer->Stop();
```

サンプルにおいてStopを実行するタイミングは、ファイルに落とす前になっています。

429 RIFFにおけるチャンクおよびID等のサイズ

●RIFFのチャンクやタグサイズからRIFFファイル内を自由に走査できるようになる　中級

参考サンプルプロジェクト　420　録音(単純版)　421　録音(完全版)

本書ですでに何回か解説している事項ですが（Tips317の図およびワンポイントとか)、ここでも再度解説しましょう。

RIFF　ID=4バイト
RIFF　データサイズ=4バイト
フォームID=4バイト

フォーマットチャンクID=4バイト
フォーマットチャンクサイズ=4バイト
フォーマットチャンクデータサイズ=sizoef(WAVEFORMATEX)　又はsizoef(WAVEFORMATEXTENSIBLE)
データチャンクID=4
デーチャンクサイズ=（波形の全サイズ）

サンプルでは数箇所「//RIFFを分かっていれば、この数字たちの意味が分かるはず」とコメントを打っている行があります。

コメント通りで、RIFFの構造を分かっていれば奇妙には見えませんが、逆に言うと分かっていなければ意味不明なマジックナンバーになるでしょう。

Tips420では143行にあります。

```
DWORD Size=4+ 8 + sizeof(g_WaveFormat)-2 + 4 +  dwSize;
```

Tips421では58行にあります。

```
DWORD Size=4+ 8 + sizeof(g_WaveFormat)-2 + 4 + dwSize;
```

Tips420の143行とTips421の58行は同じ意味です。

まずDWORD　Sizeはなにを意味するのかというと、ここではRIFFヘッダーのデータサイズに入れる値です。したがってファイルサイズ-8となる値です。最初の4は、WAVEフォームIDのサイズ4バイトです。次の8は、フォーマットチャンクのIDのサイズ4バイトとフォーマットチャンクのチャンクサイズのサイズ4バイトを足したものです。

sizeof(g_WaveFormat)-2は、WAVEFORMATEX構造体のサイズ-2です。WAVEFORMATEX構造体のサイズ-2とRIFFファイルのフォーマット部分のサイズは同じなのでこのように書けます。-2するのは、構造体側にcbSizeメンバがあり、ファイル側にはないからです。cbSizeメンバはWORD型（2バイト）ですから2を引きます。WAVEFORMATEX構造体についてもTips416で解説しています。

次の4は、データチャンク（波形データチャンク）のIDのサイズ4バイトです。dwSizeは、波形データのサイズが入っています。したがって、これらの合計はRIFFタグのサイズ4バイトとRIFFデータサイズのサイズ4バイトを除いたファイル全体のサイズ、ファイルサイズ-8となるわけです。

430 キャプチャーしたサウンドをWAVファイルにする

mmio系APIを使用せずにファイルに落とす

上級

参考サンプルプロジェクト　420 録音（単純版）　421 録音（完全版）

mmio系の関数群は便利ではありますが、一度はCランタイムの標準IO関数でコーディングすることをお勧めします。

この場合に限ったことではありませんが、“便利さ” と “本質の把握” は大抵トレードオフになるということは開発者であれば常に心に留めておくべきです。要はバランスの問題なのですが。

さて、Tips420、421共にSaveToFile関数でファイル化を行っていますが、Tips421は分かり辛いので、次のコードはTips420のほうです。

```
VOID  SaveToFile()
{
    g_DSCaptureBuffer->Stop();
```

まず、キャプチャーバッファを停止します。

```
    DWORD  dwSize=0;
    VOID*  pv=NULL;
    //単純に全体をロック
    g_DSCaptureBuffer-
>Lock(0,NULL,&pv,&dwSize,NULL,NULL,DSCBLOCK_ENTIREBUFFER);
    if(NULL==pv)
    {
          MessageBox(0,"ロック失敗","",MB_OK);
          return;
    }
```

次に、キャプチャーバッファをロックし、WAVデータを取り出せるようにします。

```
    //WAVファイル新規作成
    FILE*  fp=fopen("Rec.wav","wb");
```

標準のIO関数、fopenによりRec.wavというファイル名でファイルを書き込みモードで新規作成します。これ以降は、ファイル内にRIFFイメージを作り上げていきます。

まず最初は、RIFFチャンクヘッダーを書き込んでいきます（Tips317の図を見なが

らコードを見ると分かりやすいでしょう)。

```
	//RIFFチャンクヘッダーを書き込む
	DWORD  Fourcc=mmioStringToFOURCC("RIFF",  0);
	fwrite((const  void*)&Fourcc,4,1,fp);
	DWORD  Size=4+  8  +  sizeof(g_WaveFormat)-2  +  4  +   dwSize;//RIFFを分かっていれば、この数字たちの意味が分かるはず
	fwrite((const  void*)&Size,4,1,fp);
	DWORD  ID=mmioStringToFOURCC("WAVE",  0);
	fwrite((const  void*)&ID,4,1,fp);
```

フォーマットサブチャンクを書き込んでゆきます。

```
	//フォーマットサブチャンクを書き込む
	Fourcc=mmioStringToFOURCC("fmt  ",  0);
	fwrite((const  void*)&Fourcc,4,1,fp);
	Size=sizeof(g_WaveFormat)-2;//dbSizeはファイルには無いので-2
	fwrite((const  void*)&Size,4,1,fp);
	fwrite((const  void*)&g_WaveFormat,1,sizeof(g_WaveFormat)-2,fp);
```

データサブチャンクを書き込んでゆきます。

```
	//データサブチャンクを書き込む
	Fourcc=mmioStringToFOURCC("data",  0);
	fwrite((const  void*)&Fourcc,4,1,fp);
	fwrite((const  void*)&dwSize,4,1,fp);
	fwrite((const  void*)pv,1,dwSize,fp);
```

この時点で完了です。あとはファイルを閉じて、バッファをアンロックします。

```
	fclose(fp);
	g_DSCaptureBuffer->Unlock(pv,dwSize,NULL,NULL);
}
```

第 9 章

431～433

サウンドカードの調査の極意

9-1　サウンドカード（431～433）

431 利用可能なサウンドカードを列挙する

●PCに装備されている全サウンドカードを列挙する

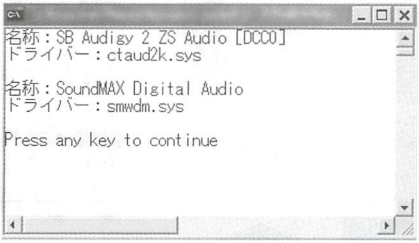
初級

サンプルプロジェクト　431　サウンドカード列挙

PCに装備されているサウンドカードを全て列挙するにはDirectSoundEnumerate関数を使用します。関数の第1引数にはコールバック関数のポインタを、第2引数にはコンテキストを渡します。

コードは単純で、次のようになります。

▼実行結果



```
名称：SB Audigy 2 ZS Audio [DCC0]
ドライバー：ctaud2k.sys

名称：SoundMAX Digital Audio
ドライバー：smwdm.sys

Press any key to continue
```

```
BOOL CALLBACK DSEnumProc(LPGUID lpGUID,LPCTSTR szName, LPCTSTR
szDriverName,VOID* pContext)
{
   if(lpGUID != NULL)
   {
        printf("名称：%s¥n",szName);
        printf("ドライバー：%s¥n¥n",szDriverName);
   }
   return TRUE;
}

VOID main()
{
     DirectSoundEnumerate( (LPDSENUMCALLBACK)DSEnumProc,NULL);
}
```

コールバック関数は、引数を好きなように利用して必要な情報を表示あるいは保存すればいいでしょう。

lpGUIDはそのサウンドデバイスのGUIDです。デバイスには必ず一意のGUIDが割り当てられています。本サンプルでは使用していません。

第2引数はデバイスの名称、第3引数はそのデバイスが使用しているドライバー名です。

第4引数はコンテキストで、ウィンドウハンドルなどを渡します。それによりコールバック関数内でそのウィンドウ上に描画したりすることができます。本サンプルには不用です。

432 サウンドカードの再生能力を調べる

●IDirectSound8:: GetCapsメソッドにより再生能力を取得する

初級

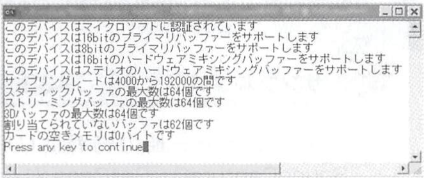
サンプルプロジェクト 432 再生能力

再生能力情報はIDirectSound8:: GetCapsメソッドにより取得します。

IDirectSound8:: GetCapsには引数としてDSCAPS型のインスタンスを渡します。関数から戻るとそのインスタンスに情報が格納されています。

サンプルのコードは次のようになっています。capsの中身を表示しているだけです。冗長になるのでコードは端折ったものを掲載します。

▼実行結果



```
このデバイスはマイクロソフトに認証されています
このデバイスは16bitのプライマリバッファーをサポートします
このデバイスは8bitのプライマリバッファーをサポートします
このデバイスは16bitのハードウェアミキシングバッファーをサポートします
このデバイスはステレオのハードウェアミキシングバッファーをサポートします
サンプリングレートは4000から192000の間です
スタティックバッファの最大数は64個です
ストリーミングバッファの最大数は64個です
3Dバッファの最大数は64個です
割り当てられていないバッファは62個です
カードの空きメモリは0バイトです
Press any key to continue
```

```
VOID main()
{
    LPDIRECTSOUND8 pDS;
    DirectSoundCreate8(NULL,&pDS,NULL);

    DSCAPS caps;
    caps.dwSize=sizeof(DSCAPS);
    pDS->GetCaps(&caps);

    if((caps.dwFlags & DSCAPS_CERTIFIED)!=0)
    {
            printf("このデバイスはマイクロソフトに認証されています¥n");
    }
(同様なので省略)

    printf("サンプリングレートは%dから%dの間です¥n",
            caps.dwMinSecondarySampleRate,
            caps.dwMaxSecondarySampleRate);

(同様なので省略)

    pDS->Release();
}
```

One Point

図中で、サウンドカードのローカルメモリが0と報告されている点については、Tips342を参照すれば理由が分かります。

433 サウンドカードのキャプチャー能力を調べる

IDirectSoundCapture8:: GetCapメソッドによりキャプチャー能力を調査する　初級

サンプルプロジェクト　433 キャプチャー能力

キャプチャー能力情報はIDirectSoundCapture8:: GetCapsメソッドにより取得します。

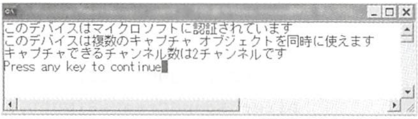
IDirectSoundCapture8:: GetCapsには引数としてDSCCAPS型のインスタンスを渡します。

DSCCAPSはIDirectSound8:: GetCapsのときの引数DSCAPSと似ていますが異なります。

IDirectSound8:: GetCap（&DSCAPS）
IDirectSoundCapture8:: GetCap（&DSCCAPS）

関数から戻るとそのインスタンスに情報が格納されています。

▼実行結果



```
このデバイスはマイクロソフトに認証されています
このデバイスは複数のキャプチャ オブジェクトを同時に使えます
キャプチャできるチャンネル数は2チャンネルです
Press any key to continue
```

サンプルのコードは次のようになっています。capsの中身を表示しているだけです。

```
VOID main()
{
    LPDIRECTSOUNDCAPTURE8 pDSCapture;
    DirectSoundCaptureCreate8(&DSDEVID_DefaultCapture,&pDSCapture,NULL);

    DSCCAPS caps;
    caps.dwSize=sizeof(DSCCAPS);
    pDSCapture->GetCaps(&caps);

    if((caps.dwFlags & DSCCAPS_CERTIFIED)!=0)
    {
        printf("このデバイスはマイクロソフトに認証されています¥n");
    }
    if((caps.dwFlags & DSCCAPS_EMULDRIVER)!=0)
    {
        printf("このデバイスのDirectSoundドライバがありません。処理は低速になります
¥n");
    }
    if((caps.dwFlags & DSCCAPS_MULTIPLECAPTURE)!=0)
    {
        printf("このデバイスは複数のキャプチャ オブジェクトを同時に使えます¥n");
    }

    printf("キャプチャできるチャンネル数は%dチャンネルです¥n",
        caps.dwChannels);

    pDSCapture->Release();
}
```

このサンプルでたった一つ、DSCCAPSのメンバ変数で表示していないメンバがあります。それはFlagsというメンバで、なぜ表示していないかと言うと、フォーマットを表す定数が24種類もあるため、かなり冗長になると思ったためです。

もし、フォーマットを表示するのであれば、次のようにすればいいだけです。

```
if((caps.dwFormats & WAVE_FORMAT_1M08)!=0)
{
    printf("11.025 kHz, Mono, 8-bitをサポートしています¥n");
}
(その他23種類のフォーマットに関しても同様)
```

フォーマットを表す定数はmmsyste.hで次のように定義されています。これらの論理和によりフォーマットが表されます。

```
#define  WAVE_INVALIDFORMAT    0x00000000   予期せぬフォーマット
#define  WAVE_FORMAT_1M08      0x00000001   11.025kHz,Mono,8-bit
#define  WAVE_FORMAT_1S08      0x00000002   11.025kHz,Stereo,8-bit
#define  WAVE_FORMAT_1M16      0x00000004   11.025kHz,Mono,16-bit
#define  WAVE_FORMAT_1S16      0x00000008   11.025kHz,Stereo,16-bit
#define  WAVE_FORMAT_2M08      0x00000010   22.05kHz,Mono,8-bit
#define  WAVE_FORMAT_2S08      0x00000020   22.05kHz,Stereo,8-bit
#define  WAVE_FORMAT_2M16      0x00000040   22.05kHz,Mono,16-bit
#define  WAVE_FORMAT_2S16      0x00000080   22.05kHz,Stereo,16-bit
#define  WAVE_FORMAT_4M08      0x00000100   44.1kHz,Mono,8-bit
#define  WAVE_FORMAT_4S08      0x00000200   44.1kHz,Stereo,8-bit
#define  WAVE_FORMAT_4M16      0x00000400   44.1kHz,Mono,16-bit
```

```
#define  WAVE_FORMAT_4S16      0x00000800   44.1kHz,Stereo,16-bit
#define  WAVE_FORMAT_44M08     0x00000100   44.1kHz,Mono,8-bit
#define  WAVE_FORMAT_44S08     0x00000200   44.1kHz,Stereo,8-bit
#define  WAVE_FORMAT_44M16     0x00000400   44.1kHz,Mono,16-bit
#define  WAVE_FORMAT_44S16     0x00000800   44.1kHz,Stereo,16-bit
#define  WAVE_FORMAT_48M08     0x00001000   48kHz,  Mono,8-bit
#define  WAVE_FORMAT_48S08     0x00002000   48kHz,  Stereo,8-bit
#define  WAVE_FORMAT_48M16     0x00004000   48kHz,  Mono,16-bit
#define  WAVE_FORMAT_48S16     0x00008000   48kHz,  Stereo,16-bit
#define  WAVE_FORMAT_96M08     0x00010000   96kHz,  Mono,8-bit
#define  WAVE_FORMAT_96S08     0x00020000   96kHz,  Stereo,8-bit
#define  WAVE_FORMAT_96M16     0x00040000   96kHz,  Mono,16-bit
#define  WAVE_FORMAT_96S16     0x00080000   96kHz,  Stereo,16-bit
```

第3部
DirectInput
の
極意

DirectX
Visual C++ 2002/2003/2005対応
▶逆引き大全◀
500の
極意

第 10 章

434〜453

入力（ユーザー→デバイス）の極意

10-1　初期化（434〜448）
10-2　入力データ取得（449〜453）

434 キーボード初期化の流れ

キーボード初期化段階で行う処理

初級

初期化は次のステップを踏みます。

①DirectInputオブジェクトの作成（DirectInput8Create関数）

②"キーボード・デバイスとして" DirectInputデバイスオブジェクトの作成（IDirectInput8::CreateDeviceメソッド）

③協調レベルの設定（IDirectInputDevice8:: SetCooperativeLevelメソッド）

初期化が終われば、デバイスオブジェクトからキーボード情報（キー押下）を取得することが出来ます。

435 マウス初期化の流れ

マウス初期化段階で行う処理

初級

初期化は次のステップを踏みます。

①DirectInputオブジェクトの作成（DirectInput8Create関数）

②"マウス・デバイスとして" DirectInputデバイスオブジェクトの作成（IDirectInput8::CreateDeviceメソッド）

③協調レベルの設定（IDirectInputDevice8:: SetCooperativeLevelメソッド）

初期化が終われば、デバイスオブジェクトからマウス情報（移動、ボタン押下）を取得することが出来ます。

436 ゲームコントローラー初期化の流れ

ゲームコントローラー初期化段階で行う処理

初級

ゲームコントローラーには3つのタイプがあります。ゲームパッド、ジョイスティック、そしてホイールコントローラー（ステアリングコントローラー、ハンドルコントローラー）です。これら3つは「ゲームコントローラー」とまとめて呼びますが、DirectInputではコード上「joystickジョイスティック」とまとめられています。

ゲームコントローラーの初期化はキーボード・マウスとは少々異なり、初期化のステップが多くなります。

①DirectInputオブジェクトの作成。
②使用可能なゲームコントローラーを列挙する。
③“ジョイスティック・デバイスとして” DirectInputDevice（デバイスオブジェクト）を作成。
④協調レベルの設定。
⑤ゲームコントローラーのプロパティを設定する。

②と⑤はキーボードやマウスでは無かった処理です。

なぜ列挙が必要なのでしょう？

キーボードやマウスは、通常1つのPCに1個ずつですよね。それに対し、ゲームコントローラーは、複数接続していることも珍しくないからです。

実際筆者は、（サンプルを作成するためですが）スティック、ハンドルコントローラー、パッドの3種類、数としては6個を同時に接続しています。列挙しなければ、どのコントローラーを使用すればいいのか、その中で特定のコントローラーをユーザーに選択させることが出来ません。

キーボードとマウスの入力はWin32メッセージでも処理できますが、ゲームコントローラーの入力はDirectInputならではでしょう。

437 DirectInputオブジェクトを作成する

DirectInput8Create関数によりDirectInputオブジェクトを作成する　初級

参考サンプルプロジェクト　449　キーボード入力

DirectInputオブジェクトの作成は、DirectInput8Create関数により行います。

サンプルでのコードは次のようにしています。

```
LPDIRECTINPUT8 pDinput=NULL;
DirectInput8Create( GetModuleHandle(NULL),DIRECTINPUT_VERSION,IID_IDirect
Input8, (VOID＊＊)&pDinput, NULL );
```

第1引数は当該アプリのインスタンスハンドルを渡します（ウィンドウハンドルではありません）。

438 キーボードのDirectInputデバイスオブジェクトを作成する

●IDirectInput8::CreateDeviceメソッドにGUID_SysKeyboardを指定してキーボードデバイスを作成する

初級

参考サンプルプロジェクト　449　キーボード入力

キーボード用デバイスオブジェクトは、IDirectInput8のメソッドIDirectInput8::CreateDeviceにより行います。

サンプルでのコードは次のようになっています。

```
LPDIRECTINPUTDEVICE8 pKeyDevice=NULL;
pDinput->CreateDevice( GUID_SysKeyboard,&pKeyDevice, NULL )
```

pDinputはDirectInputオブジェクトであり、適正に作成されているものとします。

そしてこれだけでは不十分で、次のようにキーボードのデータフォーマットをセットする必要があります。

```
pKeyDevice->SetDataFormat(  &c_dfDIKeyboard  )
```

c_dfDIKeyboardはDirectInput側で用意されている構造体なので、そのまま使用します。

439 マウスのDirectInputデバイスオブジェクトを作成する

●IDirectInput8::CreateDeviceメソッドにGUID_SysMouseを指定してマウスデバイスを作成する

初級

参考サンプルプロジェクト　450　マウス入力

マウス用デバイスオブジェクトは、IDirectInput8のメソッドIDirectInput8::CreateDeviceにより行います。

サンプルでのコードは次のようになっています。

```
LPDIRECTINPUTDEVICE8 pMouseDevice=NULL;
pDinput->CreateDevice(GUID_SysMouse,& pMouseDevice, NULL )
```

pDinputはDirectInputオブジェクトであり、適正に作成されているものとします。

そしてこれだけでは不十分で、次のようにマウスのデータフォーマットをセットする必要があります。

```
pMouseDevice ->SetDataFormat( &c_dfDIMouse2 )
```

c_dfDIMouse2はDirectInput側で用意されているマウスデータ用の構造体なので、そのまま使用します。

440 ゲームコントローラーのDirectInputデバイスオブジェクトを作成する

IDirectInput8::CreateDeviceメソッドにGUID_SysMouseを指定してジョイスティックデバイスを作成する

初級

参考サンプルプロジェクト　452 ゲームコントローラー

デバイスオブジェクトの作成そのものはキーボード・マウスと同じで、IDirectInput8::CreateDeviceで行います。サンプルでのコードを抜粋したものが次のコード片です。

```
LPDIRECTINPUTDEVICE8 pJoyDevice=NULL;
pDinput->CreateDevice(pdidInstance->guidInstance,&pJoyDevice, NULL);
```

しかし、はて？　第1引数を見てください。キーボード・マウスの時では既に定義されているGUIDを渡していましたが、ここでは何かアプリ側で用意したっぽいものを指定していますね。

Tips436でゲームコントローラーの初期化は列挙が必要だと述べました。このコード片は列挙時のコールバック関数内のものです。

```
BOOL CALLBACK EnumJoysticksCallback(const DIDEVICEINSTANCE＊ pdidInstance,
VOID＊ pContext)
{
    //複数列挙される場合、ユーザーに選択・確認させる
    TCHAR szConfirm[MAX_PATH+1];
    sprintf(szConfirm,"この物理デバイスでデバイスオブジェクトを作成しますか？¥n%s¥n%s",
        pdidInstance->tszProductName,pdidInstance->tszInstanceName);
    if(MessageBox(0,szConfirm,"確認",MB_YESNO)==IDNO)
    {
        return DIENUM_CONTINUE;
    }
    // 「DirectInputデバイス」オブジェクトの作成
    if(FAILED( pDinput->CreateDevice(pdidInstance->guidInstance,
                                     &pJoyDevice, NULL)))
    {
        return DIENUM_CONTINUE;
    }
    return DIENUM_STOP;
}
```

このコールバック関数は、仕様としては単純なほうだと思います（苦労して単純にしたんですが）。列挙される度に、ユーザーに、現在列挙されているデバイスを使用するかどうか確認します。ユーザーが確認のメッセージボックスでOKを押せば、その次の処理、つまり現在列挙されているデバイスでデバイスオブジェクトを作成します。

もしユーザーが確認ダイアログで、NOを選べば、次のデバイス列挙を試みます。

```
if(MessageBox(0,szConfirm,"確認",MB_YESNO)==IDNO)
{
    return  DIENUM_CONTINUE;←次の列挙を行うという意味でリターン。
}
```

当然ですが、デバイスが一個しか無い場合は、コールバックは一回しか呼ばれません。その場合NOを押すと、このサンプルではエラー処理を省いているので強制終了するでしょう。

デバイスを作成したら、キーボード・マウスの場合と同じようにデータフォーマットをセットします。

```
pJoyDevice->SetDataFormat(  &c_dfDIJoystick2  );
c_dfDIJoystick2はDirectInput側で用意されている構造体です。
```

441 ゲームコントローラーを列挙する

●コールバック関数を実装したうえでIDirectInput8::EnumDevicesメソッドを実行する　初級

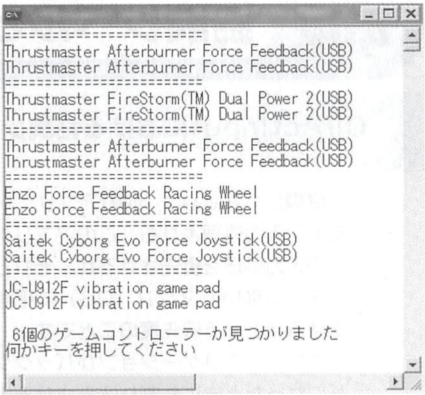
サンプルプロジェクト　441　コントローラー列挙

PCに接続されているコントローラーを列挙するにはIDirectInput8::EnumDevicesメソッドを使用します。このメソッドにはアプリケーション定義のコールバック関数を渡します。

コールバック関数はDirectInput側が実行

するものですが、実装はアプリケーション側で行います。

▼実行結果



```
=========================
Thrustmaster Afterburner Force Feedback(USB)
Thrustmaster Afterburner Force Feedback(USB)
=========================
Thrustmaster FireStorm(TM) Dual Power 2(USB)
Thrustmaster FireStorm(TM) Dual Power 2(USB)
=========================
Thrustmaster Afterburner Force Feedback(USB)
Thrustmaster Afterburner Force Feedback(USB)
=========================
Enzo Force Feedback Racing Wheel
Enzo Force Feedback Racing Wheel
=========================
Saitek Cyborg Evo Force Joystick(USB)
Saitek Cyborg Evo Force Joystick(USB)
=========================
JC-U912F vibration game pad
JC-U912F vibration game pad

 6個のゲームコントローラーが見つかりました
何かキーを押してください
```

```
//列挙のコールバック関数
BOOL CALLBACK DIEnumDevicesProc( LPCDIDEVICEINSTANCE lpddi,LPVOID pvRef)
{
      g_dwCount++;
      printf("=========================¥n");
      printf("%s¥n%s¥n",lpddi->tszInstanceName,lpddi->tszProductName);

      return DIENUM_CONTINUE;
}
```

コールバック関数の引数LPCDIDEVICEINSTANCE XXXに情報が入っているので、コールバックされる度にその中身を見れば、デバイス名等を取り出すことが出来ます。

```
LPDIRECTINPUT8  g_pDI  =  NULL;
DWORD  g_dwCount=0;
```

```
//ゲームコントローラー機器の列挙
if(FAILED(g_pDI->EnumDevices(DI8DEVCLASS_GAMECTRL,DIEnumDevicesProc,NULL,
DIEDFL_ATTACHEDONLY)))
{
      printf("列挙に失敗しました…");
      return E_FAIL;
}
```

442 協調レベルとは？

●DirectInputにおける協調レベルの意味を理解する

初級

協調レベルとは、“他のアプリケーションでの”入力処理よりも、このアプリケーションの入力処理を優先させるか、それとも、特別扱いしないでマシン（OS）に平等に処理させるかという優先度のことです。

また、アプリケーションがバックグラウンドになったときにも割り込みさせるようにするかというフラグも指定できます。表現を変えると協調レベルとはデバイスを当該アプリケーションがどの程度占有するかということです。協調レベルのフラグは、排他的か非排他的かということと、フォアグラウンドかバックグラウンドかという2種類の状態だけので組み合わせは全部で4種類しかありません。

具体的に書くと、

①DISCL_NONEXCLUSIVE ¦ DISCL_BACKGROUND　非排他でバックグラウンド
②DISCL_NONEXCLUSIVE ¦ DISCL_FOREGROUND　非排他でフォアグラウンド
③DISCL_EXCLUSIVE ¦ DISCL_BACKGROUND　排他でバッググランド
④DISCL_EXCLUSIVE ¦ DISCL_FOREGROUND　排他でフォアグラウンド

の4通りになります。

実はあと一つDISCL_NOWINKEYというフラグがあります。これはキー入力に関するウィンドウメッセージを無効にするものですが、あまり用途がないと思われるので説明から除きました。

なお、“バックグラウンド”はぱっと見ておそらく逆の意味で捉えると思います。バックグラウンドは、“バックグラウンドでも入力を取得できる”ようにするということで、フォアグランドよりも強力なモードです。

敢えて言うなら、最初のうちは排他・非排他は「非排他」にしたほうがいいかもしれない、ということぐらいでしょうか。アプリケーションがウィンドウモードである場合に排他モードにするとクライアント領域の外ではカーソルが消えることがあるからです。

最初は常にDISCL_NONEXCLUSIVE ¦ DISCL_BACKGROUND（非排他でバックグラウンド）と指定してもいいくらいです。この組み合わせはSetCooperativeLevelメソッドのデフォルト設定にもなっています。

443 協調レベルの設定

●IDirectInputDevice8::SetCooperativeLevelメソッドで協調レベルを設定する

初級

協調レベルのセットは、IDirectInputDevice8::SetCooperativeLevelメソッドで行います。

キーボードサンプルでは次のように設定し

ています。

```
pKeyDevice->SetCooperativeLevel(hWnd,DISCL_NONEXCLUSIVE | DISCL_BACKGROUND
);
```

協調レベルの意味、モードの意味が分からない場合はTips442を参照してください。

444 デバイスの取得とは？

デバイスの“取得”という概念背景を理解する　初級

デバイスの作成は分かると思いますが、「取得」とはなんのことなのでしょう？作成とは違う概念なのでしょうか？

答えを先に書くと「作成」と「取得」は異なります。

DirectInputにおいては、デバイスの「作成」と「取得」という2つの動作が定義されています。「作成」はIDirectInput8::CreateDeviceにより行いました。実はデバイスを作成しただけではデバイスにアクセスできず、入力情報を得ることができません。作成したデバイスへのアクセス権を得る必要があります。取得とはアクセス権を取得することです。

445 デバイスを取得する

IDirectInputDevice8::Acquireメソッドでアクセス権を取得する　初級

当該デバイスへのアクセス権を取得するには、IDirectInputDevice8::Acquireメソッドを使用します。初期化時、デバイスを作成した直後にでも取得を行います。

また、DirectInputではアクセス権は常にあるものではなく、一般的にアプリケーションは実行中に何度もアクセス権を失う可能性があります。

アプリケーションがアクセス権を失ったときには、Acquireメソッドを実行してデバイスを再取得する必要があります。

446 キーボード初期化

●キーボードから入力を取得できるように初期化する

初級

参考サンプルプロジェクト　449　キーボード入力

コードは次のようになります。ソース上のコメント通りなので、解説は割愛します。

```
// 「DirectInput」オブジェクトの作成
DirectInput8Create( GetModuleHandle(NULL),DIRECTINPUT_VERSION,IID_IDirect
Input8, (VOID＊＊)&pDinput, NULL )；
// 「DirectInputデバイス」オブジェクトの作成
pDinput->CreateDevice( GUID_SysKeyboard,&pKeyDevice, NULL )；
// デバイスをキーボードに設定
pKeyDevice->SetDataFormat( &c_dfDIKeyboard )；
// 協調レベルの設定
pKeyDevice->SetCooperativeLevel(hWnd,DISCL_NONEXCLUSIVE | DISCL_BACKGROUND
)；
// デバイスを「取得」する
pKeyDevice->Acquire();
```

447 マウス初期化

●マウスから入力を取得できるように初期化する

初級

参考サンプルプロジェクト　450　マウス入力

コードは次のようになります。ソース上のコメント通りなので、解説は割愛します。

```
LPDIRECTINPUT8  pDinput=NULL;
LPDIRECTINPUTDEVICE8  pJoyDevice=NULL;
```

```
//　「DirectInput」オブジェクトの作成
DirectInput8Create( GetModuleHandle(NULL),DIRECTINPUT_VERSION,IID_IDirect
Input8, (VOID＊＊)&pDinput, NULL );
//　「DirectInputデバイス」オブジェクトの作成
pDinput->CreateDevice( GUID_SysMouse,&pMouseDevice, NULL );
// デバイスをマウスに設定
pMouseDevice->SetDataFormat( &c_dfDIMouse2 );
// 協調レベルの設定
pMouseDevice->SetCooperativeLevel(hWnd,DISCL_NONEXCLUSIVE |
DISCL_FOREGROUND );
// デバイスを「取得」する
pMouseDevice->Acquire();
```

448 ゲームコントローラー初期化

ゲームコントローラーから入力を取得できるように初期化する

初級

参考サンプルプロジェクト　452 ゲームコントローラー

ゲームコントローラーにおける初期化の流れをもう一度書きます。

①DirectInputオブジェクトの作成。

②使用可能なゲームコントローラーを列挙する。

③“ジョイスティック・デバイスとして”DirectInputDevice（デバイスオブジェクト）を作成。

④協調レベルの設定。

⑤ゲームコントローラーのプロパティを設定する。

コードは次のようになります。

```
// 「DirectInput」オブジェクトの作成
DirectInput8Create( GetModuleHandle(NULL),
    DIRECTINPUT_VERSION,IID_IDirectInput8, (VOID＊＊)&pDinput, NULL ) ) )
 //利用可能なゲームコントローラーの列挙関数を実行
pDinput->EnumDevices(DI8DEVCLASS_GAMECTRL, EnumJoysticksCallback,
            NULL, DIEDFL_ATTACHEDONLY)))
// デバイスをジョイスティックに設定
pJoyDevice->SetDataFormat( &c_dfDIJoystick2 );
// 協調レベルの設定
pJoyDevice->SetCooperativeLevel(hWnd,DISCL_NONEXCLUSIVE | DISCL_BACKGROUND
);
//アプリケーションで使用するコントローラーのプロパティを列挙して設定する
```

```
pJoyDevice->EnumObjects( EnumObjectsCallback,NULL, DIDFT_ALL );
// デバイスを「取得」する
pJoyDevice->Acquire();
```

まず、EnumDevicesは何のために行っているのかと言うと、先にも述べましたが、利用可能なデバイスを列挙しています。

次はEnumObjectsですが、これは列挙されたコントローラーのプロパティを設定しています。

この2つのメソッド用にそれぞれにコールバック関数を用意しなくてはなりません。それぞれのコールバック関数は、それぞれの目的を実現するような実装になっています。

DirectInputの基本処理コード全般に言えますが、このコールバック関数も定型的なコーディングであり、深く考える必要はあまりなく、また、そのまま別のプロジェクトでも使用できると思います。

●列挙用のコールバック関数

```
BOOL CALLBACK EnumJoysticksCallback(const DIDEVICEINSTANCE* pdidInstance,
VOID* pContext)
{
     //複数列挙される場合、ユーザーに選択・確認させる
     TCHAR szConfirm[MAX_PATH+1];
     sprintf(szConfirm,"この物理デバイスでデバイスオブジェクトを作成しますか？¥n%s¥n%s",
          pdidInstance->tszProductName,pdidInstance->tszInstanceName);
     if(MessageBox(0,szConfirm,"確認",MB_YESNO)==IDNO)
     {
          return DIENUM_CONTINUE;
     }
     // 「DirectInputデバイス」オブジェクトの作成
     if(FAILED( pDinput->CreateDevice(pdidInstance->guidInstance,
                                        &pJoyDevice, NULL)))
     {
          return DIENUM_CONTINUE;
     }
     return DIENUM_STOP;
}
```

●プロパティ設定用のコールバック関数

```
BOOL CALLBACK EnumObjectsCallback( const DIDEVICEOBJECTINSTANCE* pdidoi,VOID
* pContext )
{

     if( pdidoi->dwType & DIDFT_AXIS )
     {
        DIPROPRANGE diprg;
        diprg.diph.dwSize       = sizeof(DIPROPRANGE);
        diprg.diph.dwHeaderSize = sizeof(DIPROPHEADER);
        diprg.diph.dwHow        = DIPH_BYID;
        diprg.diph.dwObj        = pdidoi->dwType;
        diprg.lMin              = -MOVE_SPEED;
        diprg.lMax              = MOVE_SPEED;

        if( FAILED( pJoyDevice->SetProperty( DIPROP_RANGE, &diprg.diph ) ) )
          {
                 return DIENUM_STOP;
          }
    }
    return DIENUM_CONTINUE;
}
```

449 押下されているキー(キーボード)を取得する

●IDirectInputDevice8::GetDeviceStateメソッドによりキーボードデータを取得する 初級

サンプルプロジェクト 449 キーボード入力

初期化が終了すればキーボードの状態を得ることが出来ます。キーの状態はIDirectInputDevice8::GetDeviceStateメソッドにより取得します。

そして通常は、その前にIDirectInputDevice8::Acquireメソッドも実行します。Acquireの目的はフォーカスを失うなどしてアクセス権を失っている場合に対処するためです（Tips444）。

irectInputDevice8::GetDeviceStateメソッドには引数として、BYTE型の配列を渡します。メソッドから戻ると、その配列にキーボードの全部のキーについての情報が格納されていますので、それを好きなように参照して利用できます。

コードは次のようになります。

```
VOID KeyInput()
{
    HRESULT hr=pKeyDevice->Acquire();
    if((hr==DI_OK) || (hr==S_FALSE))
    {
          BYTE diks[256];
          pKeyDevice->GetDeviceState(sizeof(diks),&diks);
```

配列の要素数は256個と決まっています。256個以上キーがあるキーボードは無いので十分なサイズです。キーの一つ一つが1つの要素に対応します。

サンプルでは、矢印キー（DIK_LEFT～DIK_DOWN）が押されている場合に座標（fPosX、fPosY）を適当に増減させています。

```
           if(diks[DIK_LEFT] & 0x80)
           {
                  fPosX-=4;
           }
           if(diks[DIK_RIGHT] & 0x80)
           {
                  fPosX+=4;
           }
           if(diks[DIK_UP] & 0x80)
           {
                  fPosY-=4;
           }
           if(diks[DIK_DOWN] & 0x80)
           {
                  fPosY+=4;
           }
    }
}
```

DIK_で始まる定数はDirectInput側で用意（定義）されているものなのでそれを使用します。どのキーがどの定数に割り当てられているかは、だいたい名前から分かると思います。

450 マウスの動きを取得する

IDirectInputDevice8::GetDeviceStateメソッドによりマウスデータを取得する　初級

サンプルプロジェクト　450　マウス入力

初期化が終了すればマウスの状態を得ることが出来ます。マウスの状態はIDirectInputDevice8::GetDeviceStateメソッドにより取得します。そして通常は、その前にIDirectInputDevice8::Acquireメソッドも実行します。Acquireの目的はフォーカスを失うなどしてアクセス権を失っている場合に対処するためです（Tips444）。

IDirectInputDevice8::GetDeviceStateメソッドには引数として、DIMOUSESTATE2型のインスタンス１つのアドレスを渡します。メソッドから戻ると、その構造体にマウスについての情報が格納されていますので、それを好きなように参照して利用できます。

コードは次のようになります。

```
HRESULT  MouseInput()
{
    DIMOUSESTATE2  dims={0};
    if(FAILED(pMouseDevice->GetDeviceState(  sizeof(DIMOUSESTATE2),  &dims
)))
    {
          pMouseDevice->Acquire();
          return  E_FAIL;
    }
```

サンプルでは、dimsに入っているマウス情報をにより座標（fPosX、fPosY）を適当に増減させています。

```
pDevice->SetRenderState(D3DRS_SHADEMODE,D3DSHADE_FLAT);
```

```
    //カーソルの増減値をスプライトの位置に利用
    fPosX+=dims.lX;
    fPosY+=dims.lY;

    if(fPosX<0)
    {
          fPosX=0;
    }
    else  if(fPosX>WINDOW_WIDTH)
    {
          fPosX=WINDOW_WIDTH;
    }
    if(fPosY<0)
    {
          fPosY=0;
    }
    else  if(fPosY>WINDOW_HEIGHT)
    {
          fPosY=WINDOW_HEIGHT;
    }
fPosX+=dims.lX;
fPosY+=dims.lY;
```

マウスの位置は、dims構造体のlXとlYメンバに格納されていますので、それを利用します。厳密に言うと、直前の位置からの相対距離（ピクセル単位）、つまり、「何ピクセル移動したか」という増分です。

451 マウスボタンの押下を取得する

●IDirectInputDevice8::GetDeviceStateメソッドによりマウスデータを取得する　初級

サンプルプロジェクト　450　マウス入力

サンプルでは、dimsに入っているマウス情報をにより座標のほかに、回転（fAngle）、スケール（vec3Scale）も適当に増減させています。

```
HRESULT MouseInput()
{
    DIMOUSESTATE2 dims={0};
    if(FAILED(pMouseDevice->GetDeviceState( sizeof(DIMOUSESTATE2), &dims )))
    {
           pMouseDevice->Acquire();
           returnE_FAIL;
    }
```

```
    //左右ボタンをスプライトのスケールに利用
    if(dims.rgbButtons[0] & 0x80)
    {
           vec3Scale＊=0.99;
    }
    if(dims.rgbButtons[1] & 0x80)
    {
           vec3Scale＊=1.01;
    }
    //真ん中ボタンをスプライトの回転に利用
    if(dims.rgbButtons[2] & 0x80)
    {
           fAngle+=0.1;
    }
    return S_OK;
}
```

リスト中のif(dims.rgbButtons[0] & 0x80)は、rgbButtonsはボタンの押下を格納している配列のポインターです。要素数は8つまであります。すなわち、8つのボタンまで対応できます。最近のマウスでは8つボタンも有り得るので、必要な要素数でしょう。ただし、一般的には3つボタンが圧倒的に多いとは思いますので、[2]以上の要素を調べる機会はあまりないでしょう。

0x80との論理積が真であれば、そのボタンが押されていることを意味します。

マウスの3つのボタンをそれぞれ調べて、押されていれば拡大と縮小、そして回転を行っています。

452 ゲームコントローラーのユーザー操作を取得する

IDirectInputDevice8::GetDeviceStateメソッドによりコントローラー状態を取得する

初級

サンプルプロジェクト 452 ゲームコントローラー

キーボードの場合、状態データを受け取るアプリケーション側の“受け皿”はBYTE型の配列、マウスの場合はDIMOUSESTATE2構造体、

そして、ゲームコントローラーの場合、受け皿はDIJOYSTATE2構造体になります。

```
DIJOYSTATE2  js={0};
```

DIJOYSTATE2構造体型のインスタンスjsを宣言し、同時にメンバ全てをゼロで初期化しています。

```
HRESULT  hr=pJoyDevice->Acquire();
pJoyDevice->GetDeviceState(sizeof(DIJOYSTATE2),  &js);;
```

キーボードやマウスでと同じように、IDirectInputDevice8::GetDeviceStateメソッドによりデータを取得します。このメソッドが成功した時点で、jsには現在のゲームコントローラーの状態が格納されています。

あとは、jsの各メンバを調べて、移動、回転、スケーリングに利用しています。それについてはキーボードやマウスと同様なので解説は割愛します。

453 アクセス権の消失に対処する

●IDirectInputDevice8::Acquireメソッドによりアクセス権を再取得する　初級

すでに解説していることでありますが、改めて解説します。

アクセス権の消失とはなにか？

アプリケーションがデバイス（キーボード、マウス、コントローラー）にアクセスできなくなることです。

消失するとどうなるか？

デバイス（キーボード、マウス、コントローラー）のデータを取得できなくなります。

どのような時に消失が起こるのか？

フォーカスが他のアプリケーションやデスクトップに移るなどして、当該アプリがアクティブでなくなったときです。

消失した場合は、IDirectInputDevice8::Acquireメソッドを実行して"再取得"しなくてはなりません。

確実なのは、データを取得する直前にでもAcquireすることです。したがって、毎フレーム再取得することになりますが、不具合はありません。

DirectX

500 Tips to Use DirectX Better!

第 11 章

454～485

フォースフィードバック（デバイス→ユーザー）の極意

11-1　FFBの基本と共通事項（454～463）
11-2　能動的フォース（ユーザー入力非感応タイプ）（464～476）
11-3　受動的フォース（ユーザー入力感応タイプ）（477～483）
11-4　FFB機器の調査（484～485）

454 フォースフィードバックとは？

力を返す技術

初級

フォースフィードバック（Force FeedBack以降FFBと表記）は、その名の通りフォース（力）を、デバイス側からフィードバック（返す）する技術です。フィードバックされた力をプレイヤーに伝えるのはジョイスティックなどのコントローラーです。

例えば、ゲーム中でマシンガンを撃った時、ガガガガツ…という発射音と共に、コントローラーもブルブルと小刻みに振動します。あるいはゲーム内で魚を釣った時、魚が糸を引く動作にシンクロしてスティックも引っ張られます。また、ホイール（ハンドル）タイプのコントローラーにおいて、砂漠を走行しているときに路面から強力な反力を受け、ハンドルの重さが増します。

FFBによって、ゲームはより一層リアルさを増すことは間違いないでしょう。FFBスティックを2本使って、3Dシーン中で“手”を表現すると面白いのではと筆者は思っていて、現在開発中のゲームにはそのようなフィーチャーも盛り込もうと思っているくらいです。

455 バイブレーションとフォースフィードバック

単純振動とFFBの違いを把握する

初級

ゲームパッドでのFFBは単純な振動によるバイブレーションです。PCから何らかの力を返すという点を見れば、フォースフィードバックとも言えるかもしれませんが、プログラムから操作できるのはその強弱だけなので、とてもFFBとは言えません。

FFBとは、もっときめ細かくリアルな動作を行うことが出来るものです。強弱だけではなく、方向、周期、そしてユーザーの力に対応した力、すなわち反力を発生させるアクチュエーター（フォースを発生させるモーター等）を介在するもので、単純なバイブレーターでは物理的に実現できず、それらをFFBと呼ぶには疑問が残ります。ただ、コード上はFFBとして運用します。

456 マグニチュードとは？

●「フォースの強さ」を意味する「単位」

初級

マグニチュードとは「フォースの強さ」を意味する「単位」です。

一般的にはマグニチュードは地震エネルギーの単位です。だからといって、地震と関連しているかと言えばそうでもなく、なぜマグニチュードと呼ぶのかは正直不明です。

ただ、“フォース”と表現しているのであれば“マグニチュード”が適当だとは思います。一般的にフォースの単位はマグニチュードですから。

そして、（DirectInputに限らず）そもそもプログラム上の単位などというものは相対的な論理値ですから、平たく言えば「何でもいい」わけです。例えばDirect3Dでも、1単位を1メートルとするか1ミリとするかは、その時のアプリケーションの単位系でどうとでもなることですから。

FFBの世界では、単位はマグニチュードが一般的なので、それに従いましょう。

457 DirectInput時間の単位は？

●FFBにおける時間の単位はマイクロ秒

初級

DirectInputのFFBにおける時間の単位はマイクロ秒です。表現を変えると100万分の1秒です（なおミリ秒は千分の1秒）。

したがって、例えば1秒間フォースをかける場合は1,000,000と指定します。

また、永遠にフォースをかけるためにINFINITEという定数が用意されています。INFINITEは0xffffffffと定義されていますが、DirectInputは0xffffffffと指定されると永遠とみなして処理します。

458 ディレクションとは？

●フォースの方向のこと

初級

ディレクション（Direction：方向）とは、フォースの方向のことです。FFBではフォースに方向性を持たせることができます。力の絶対量は同じでもディレクション（方向）が変化すれば、よりリアルと感じるわけです。

例えば、同じ力でも左に引っ張られる、右に引っ張られるという状態はディレクションを反対に変化させることにより実現します。

459 エンベロープとは？

●フォース・時間曲線グラフの包絡線

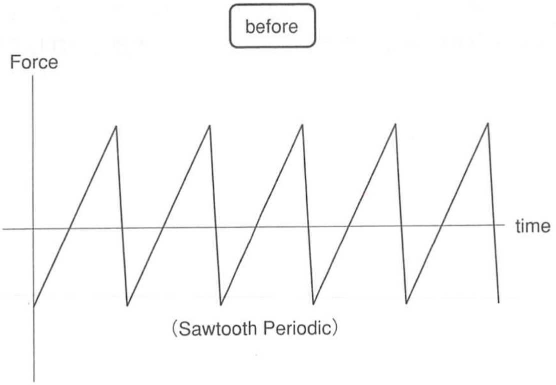
初級

エンベロープ（Envelope：包絡線）は、文字通り何らかのグラフのカーブ、エッジ、波形の外形、つまり包絡線です。ここではフォース・時間曲線グラフの包絡線です。

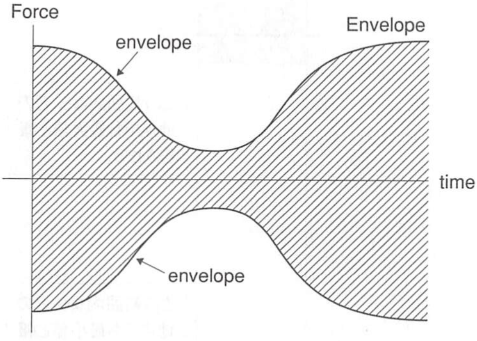
DirectSoundでもエンベロープが似たような意味で出てきます。音圧波形グラフにおける波形の全体“形状”がそうです。

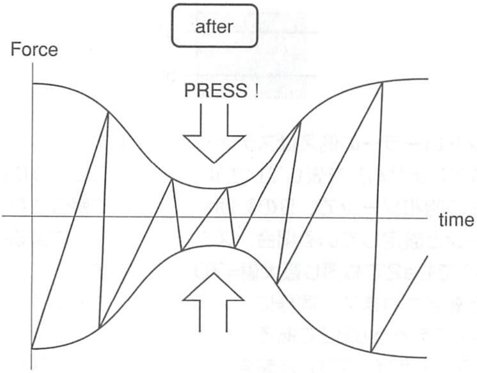
FFBでも意味は同じですが正確には、エンベロープは波形（力の波形）に対する付加的な変化を意味します。

例えば、フォースのサイン波やノコギリ波は力の時間的変化をグラフに表したときに現われるグラフの“形状”ですよね。エンベロープはその“形状”にさらに変化をかけるものです。つまり“全体形状”レベルでの時間的変化です。



（Sawtooth Periodic）


Force
envelope
Envelope
time
envelope


after
Force
PRESS !
time

なお、エンベロープはその機器のドライバーが対応していない場合もあり、またフォースのタイプによってそもそもエンベロープを指定できないものもあります

460 サチュレイションとは？

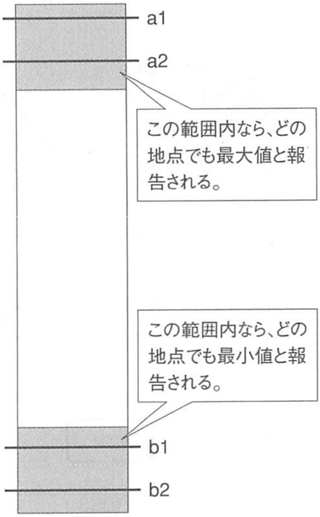
値が変わらない許容範囲という意味の“遊び”

初級

サチュレイション（Saturation：飽和）は、字面からはどのような概念なのかは想像できないことと思います。

FFBにおいて（コントローラーにおいて）飽和ゾーンとは、上限（および下限）における“遊び”のようなものです。“遊び”とは、「ハンドルの遊び」などと表現する場合の遊びのことで、要するに値が変わらない何らかの許容範囲のことです。

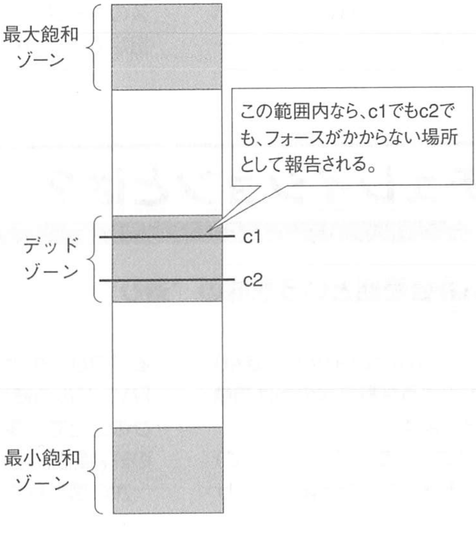
次の図を見てください。

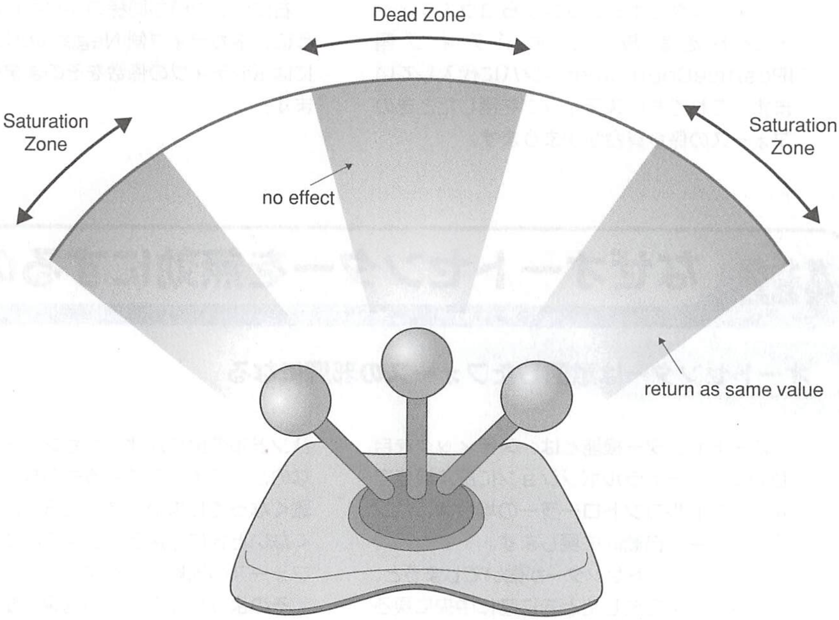

a1
a2
この範囲内なら、どの地点でも最大値と報告される。
この範囲内なら、どの地点でも最小値と報告される。
b1
b2

この図はコントローラーの例えばスティック位置（軸は特定しません）を表しています。Aの範囲は上限の飽和ゾーンで、Bの範囲が下限の飽和ゾーンと設定している場合、スティック位置がa1でもa2でも同じ最大値=70（例えば70）と報告されます。同様に、スティック位置がb1であろうがb2であろうが同じ最小値=−70（例えば−70）と報告されます。

このように飽和ゾーン内では全て同じ値として報告されます。飽和ゾーンの目的は、デバイスによる限界位置の誤差を解消することです。

このように上限と下限の“遊び”は飽和ゾーンと呼びまずが、中心の“遊び”はデッドゾーンと呼びます。


最大飽和ゾーン
この範囲内なら、c1でもc2でも、フォースがかからない場所として報告される。
デッドゾーン
c1
c2
最小飽和ゾーン


Dead Zone
Saturation Zone
Saturation Zone
no effect
return as same value

461 コウエフィシェントとは？

●フォースにかける係数のこと

初級

コウエフィシェント（Coefficient：係数）も分かりにくい概念です。係数と言っても何に対しての係数なのでしょう？

コウエフィシェントは、フォースに掛ける係数です。したがってコウエフィシェントが大きければフォースが強く、逆に小さければフォースが弱くなります。

コウエフィシェントにはプラス方向とマイナス方向があります。それぞれポジティブ・コウエフィシェント、ネガティブ・コウエフィシェントと呼びます。コントローラーによっては左側がポジティブ、右がネガティブと対応させていたり、その逆もあるようです。

次の例はスプリングフォースのコウエフィシェントを計算し、設定しているコード片です。

```
DICONDITION  spring={0};
spring.lPositiveCoefficient=  sqrt(  (double)g_iXForce  *  (double)g_iXForce
+
    (double)g_iYForce  *  (double)g_iYForce  );
spring.lNegativeCoefficient=  spring.dwPositiveSaturation;
```

スティックのポジションからコウエフィシェントを計算し、ポジティブ側lPositiveCoefficientメンバに代入しています。これで左にスティックを倒したときのフォースの係り具合が決まります。

右に倒した時も同様のかかる具合になるように、ネガティブ側lNegativeCoefficientにはポジティブの係数をそのまま代入しています。

462　なぜオートセンターを無効にするの？

●オートセンターは意図したフォースの邪魔になる

初級

オートセンター機能とは、スティックを自動的にニュートラルポジションに戻す機能です。ハンドルコントローラーの場合は、ハンドルを中央に自動的に戻します。

しかし、オートセンターが働いてしまうと、ユーザーが手を放したときに常に中央に戻ろうとする力がかかってしまい、スティックやハンドルの戻りがオートセンターによるものなのか、フォースによるものなのかが分かり難くなってしまいます。さらに、戻って欲しくないときにも戻ろうとするので、意図したフォースの邪魔になります。

そのような理由でFFB効果を働かせる時には、オートセンター機能を無効にします。

463　フォースの種類について

●フォースには4種類ある

初級

フォースには種類があります。DirectInputに実装されいてるのは次の4種類です。

①コンスタント・フォース（Constant Force　一定フォース）
②ランプ・フォース（Ramp Force　傾斜フォース）
③ピリオディック・フォース（Periodic Force　周期フォース）
④コンディション・フォース（Condition Force　条件フォース）

③と④には、さらに数種のパターンが用意されています。

ピリオディックフォースには、周期のパターンが5種類用意されていまず。

①スクエア（Square　矩形波）
②サイン（Sine　サイン波）
③トライアングル（Triangle　三角波）
④ソウトゥース・アップ（Sawtooth Up　上昇ノコギリ波）
⑤ソウトゥース・ダウン（Sawtooth Down下降ノコギリ波）

コンディションフォースには次の4つのパターンがあります。

①スプリング（Spring　ばね）
②フリクション（Friction　摩擦）
③ダンパー（Damper　緩衝器）
⑤イナーシャ（Inertia　慣性）

それぞれの特徴と操作については、Tips467以降で解説しています。

コンスタント（一定）フォース

464 コンスタントフォースとは？

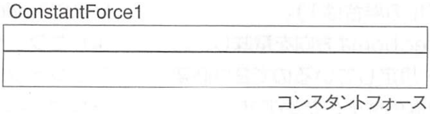
一定（コンスタント）な力を発生するもの

初級

コンスタントフォースは、一定（コンスタント）な力を発生するものです。調整（変更）できるものは“力（フォース）の強さ”しかありません。

例えば、スティックコントローラーの場合、左に2000と上に3000の力をかけて左上方向の力を発生させるなどとします。

力のかかり方が容易に想像できるので、コンスタントフォースは最も単純です。



465 コンスタントフォースを初期化する

DIEFFECT構造体をコンスタントとして初期化してエフェクトを作成する

初級

参考サンプルプロジェクト　466 コンスタントフォース

各フォースは、DirectInput上では“エフェクト”という概念で捉えます。そのエフェクトのタイプ（コンスタント、ランプ…etc）によってパラメーターが微妙に異なります。

パラメーターは、DIEFFECT構造体のメンバを埋めることによって指定します。

```
DWORD  rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG  rglDirection[2]={0,0};
DICONSTANTFORCE  cf={0};
```

```
DIEFFECT  eff;
ZeroMemory(  &eff,  sizeof(eff)  );
eff.dwSize                                         =  sizeof(DIEFFECT);
eff.dwFlags                                  =  DIEFF_POLAR  |  DIEFF_OBJECTOFFSETS;
eff.dwDuration                                  =  INFINITE;
eff.dwSamplePeriod                           =  0;
eff.dwGain                                         =  DI_FFNOMINALMAX;
eff.dwTriggerButton                          =  DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval  =  0;
eff.cAxes                                            =  g_dwFFBAxisAmt;
eff.rgdwAxes                                      =  rgdwAxes;
eff.rglDirection                              =  rglDirection;
eff.lpEnvelope                                  =  0;
eff.cbTypeSpecificParams          =  sizeof(DICONSTANTFORCE);
eff.lpvTypeSpecificParams        =  &cf;
eff.dwStartDelay                              =  0;
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

cfはコンスタントフォース用の構造体です。実はメンバを一つしか持ちません。そのメンバはマグニチュードを表します。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。ここではコンスタントフォースのエフェクトインスタンスを作成するので、第1引数にコンスタントフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(  GUID_ConstantForce,&eff,  &g_pEffect,  NULL  );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8    g_pDIDevice  =  NULL;
LPDIRECTINPUTEFFECT      g_pEffect      =  NULL;
```

466 コンスタントフォースをかける

●DICONSTANTFORCE構造体に意図した値を入れてフォースを発生させる　初級

サンプルプロジェクト　466 コンスタントフォース

本書のFFBサンプルは初期化の時にSetParameterメソッドにDIEP_STARTフラグを指定しているため、初期化後すぐにエフェクト（FFB）がかかるようになっています。ただ、初期化時はそれぞれのエフェクトタイプの構造体をゼロ初期化しているのでフォースは発生しません。

フォースが発生するのは、ユーザーがスライダーを動かしてからこのUpdateEffect関数が呼ばれたときです。

```
HRESULT UpdateEffect()
{
    //デバイスを有効にする。(有効にしないとエフェクト（FFB）が効かない)
    g_pDIDevice->Acquire();
    //エフェクトのパラメーターを設定する。更新したいパラメーターのみを指定すれば良い
    LONG rglDirection[2]={g_iXForce,g_iYForce};
    DICONSTANTFORCE cf;
    cf.lMagnitude = sqrt( (double)g_iXForce * (double)g_iXForce +
                          (double)g_iYForce * (double)g_iYForce );

    DIEFFECT eff;
    ZeroMemory( &eff, sizeof(eff) );
    eff.dwSize                  = sizeof(DIEFFECT);
    eff.dwFlags                 = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
    eff.cAxes                   = g_dwFFBAxisAmt;
    eff.rglDirection            = rglDirection;
    eff.cbTypeSpecificParams    = sizeof(DICONSTANTFORCE);
    eff.lpvTypeSpecificParams = &cf;

    if(FAILED(g_pEffect->SetParameters( &eff, DIEP_DIRECTION |
DIEP_TYPESPECIFICPARAMS |
                                 DIEP_START )))
    {
        return E_FAIL;
    }
    return S_OK;
}
```

コードは、初期化と同じであることに気付くでしょうか。変更すべきパラメーターは初期化時より少ないのでコードも初期化時より少なくて済みます。

なおg_iXForce、g_iYForceはINT型のグローバル変数で、ユーザーがダイアログのスライダーを動かしたときに増減する変数です。

ランプ（傾斜）フォース

467 ランプフォースとは？

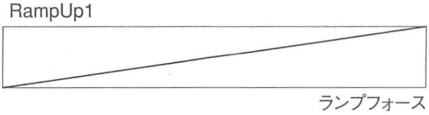
●だんだんと増加（減少）する力を発生させるもの　初級

ランプフォースは、だんだんと増加する（あるいは減少する）力を発生させるものです。

力を時間軸で表したグラフを書くと傾斜（ランプ）するので、ランプフォースというわけです。

力の強さだけではなく、その傾斜具合も設定できるので、当然コンスタントフォースに比べてパラメーターは若干多くあります。

例えば、最初は軽かったハンドルやスティックがだんだん重くなったり、逆に、重かったハンドルやスティックが緩やかに軽くなるなどというときに使用します。



468 ランプフォースを初期化する

●DIEFFECT構造体をランプとして初期化してエフェクトを作成する　初級

参考サンプルプロジェクト　469　ランプフォース

各フォースは、DirectInput上では“エフェクト”という概念で捉えます。そのエフェクトのタイプ（コンスタント、ランプ…etc）によってパラメーターが微妙に異なります。

パラメーターは、DIEFFECT構造体のメンバを埋めることによって指定します。

```
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};
DIRAMPFORCE ramp={0};
//エフェクトを作成 エフェクトはランプ（傾斜）フォースとして初期化する
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                    = sizeof(DIEFFECT);
eff.dwFlags                   = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
eff.dwDuration                 = 3000000;//マイクロ秒（ミリ秒ではない） ここでは3秒と
指定 RampにINFINITは指定出来ない
eff.dwSamplePeriod            = 0;
eff.dwGain                    = DI_FFNOMINALMAX;
eff.dwTriggerButton           = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval   = 0;
eff.cAxes                     = g_dwFFBAxisAmt;
eff.rgdwAxes                  = rgdwAxes;
eff.rglDirection              = rglDirection;
eff.lpEnvelope                = 0;
eff.cbTypeSpecificParams      = sizeof(DIRAMPFORCE);
eff.lpvTypeSpecificParams     = &ramp;
eff.dwStartDelay              = 10;
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

rampはランプフォース用の構造体です。DIRAMPFORCE ramp={0};としてゼロ初期化しているので、初期化直後はフォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここではランプフォースのエフェクトインスタンスを作成するので、第1引数にランプフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(  GUID_RampForce,&eff,  &g_pEffect,  NULL  );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8  g_pDIDevice = NULL;
LPDIRECTINPUTEFFECT   g_pEffect   = NULL;
```

469 ランプフォースをかける

●DIRAMPFORCE構造体に意図した値を入れてフォースを発生させる　初級

サンプルプロジェクト　469 ランプフォース

本書のFFBサンプルは初期化の時にSetParameterメソッドにDIEP_STARTフラグを指定しているため、初期化後すぐにエフェクト（FFB）がかかるようになっています。ただ、初期化時はそれぞれのエフェクトタイプの構造体をゼロ初期化しているのでフォースは発生しません。

フォースが発生するのはユーザーがスライダーを動かしてからこのUpdateEffect関数が呼ばれたときです。

```
HRESULT UpdateEffect()
{
    //デバイスを有効にする。(有効にしないとエフェクト（FFB）が効かない)
    g_pDIDevice->Acquire();
    //エフェクトのパラメーターを設定する。更新したいパラメーターのみを指定すれば良い
    LONG rglDirection[2]={g_iXForce,g_iYForce};
    DIRAMPFORCE ramp;
    ramp.lStart=0;
    ramp.lEnd=sqrt( (double)g_iXForce * (double)g_iXForce +
                                   (double)g_iYForce * (double)g_iYForce );

    DIEFFECT eff;
    ZeroMemory( &eff, sizeof(eff) );
    eff.dwSize                = sizeof(DIEFFECT);
    eff.dwFlags               = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
    eff.cAxes                 = g_dwFFBAxisAmt;
    eff.rglDirection          = rglDirection;

    eff.cbTypeSpecificParams  = sizeof(DIRAMPFORCE);
    eff.lpvTypeSpecificParams = &ramp;

     if(FAILED(g_pEffect->SetParameters( &eff, DIEP_DIRECTION |
DIEP_TYPESPECIFICPARAMS |
                                           DIEP_START )))
    {
          return E_FAIL;
    }
    return S_OK;
}
```

　コードは初期化と同じであるおとに気付くでしょうか。変更すべきパラメーターは初期化時より少ないのでコードも初期化時より少なくて済みます。

　なおg_iXForce、g_iYForceはINT型のグローバル変数で、ユーザーがダイアログのスライダーを動かしたときに増減する変数です。

ピリオディック（周期）フォース

470　ピリオディックフォースとは？

●周期的（ピリオディック）に力の強さが変化するフォース

初級

　ピリオディックフォースは、周期的（ピリオディック）に力の強さが変化するフォースを発生させます。

　ピリオディックフォースには、周期のパターンが5種類用意されていて、5種類の波形パターンにそれぞれ、力の強さの最小・最大値、周期の間隔、波形の位相、振幅の中心を指定できます。

　ピリオディックフォースの用途としては、爆発・怪獣の歩行時の振動など（緩やかな周期で）、あるいは、何らかの微振動・マシンガンの反動など（細かい周期で）を表現できます。

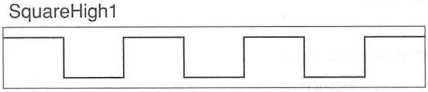
471　ピリオディックフォースのパターンについて

●5種類の波形パターンがある

初級

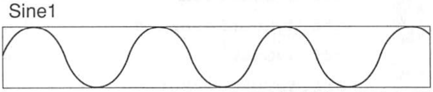
　5種類の波形パターンは次のものがあります。横軸が時間、縦軸が力です。

▼矩形波



ピリオディックフォース
スクエア

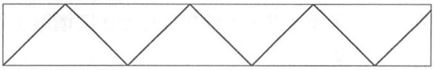
▼サイン波



ピリオディックフォース
サイン

▼三角波

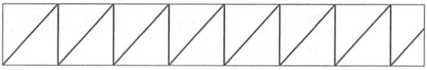
TriangleUp1



ピリオディックフォース
トライアングル

▼ノコギリ刃波形（上昇）

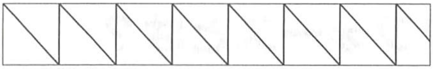
SawtoothUp1



ピリオディックフォース
ソウトゥース（アップ）

▼ノコギリ刃波形（下降）

SawtoothDown1



ピリオディックフォース
ソウトゥース（ダウン）

472 スクエア（矩形波）パターンの初期化

DIEFFECT構造体をピリオディックとして初期化してエフェクトを作成してから、スクエアパターンを指定する

初級

参考サンプルプロジェクト　476　ピリオディックフォース

スクエアパターンのピリオディックフォースの初期化は、例えば次のようになります。

```
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};

DIPERIODIC periodic={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                  = sizeof(DIEFFECT);
eff.dwFlags                 = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
eff.dwDuration              = INFINITE;
eff.dwSamplePeriod          = 0;
eff.dwGain                  = DI_FFNOMINALMAX;
eff.dwTriggerButton         = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                   = g_dwFFBAxisAmt;
eff.rgdwAxes                = rgdwAxes;
eff.rglDirection            = rglDirection;
eff.lpEnvelope              = 0;
eff.cbTypeSpecificParams     = sizeof(DIPERIODIC);
eff.lpvTypeSpecificParams    = &periodic;
eff.dwStartDelay             = 10;
g_pDIDevice->CreateEffect(GUID_Square,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

periodicはピリオディック用の構造体です。DIPERIODIC　periodic={0};としてゼロ初期化しているので、初期化直後はフォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（ピリオディックフォースの）矩形波フォースのエフェクトインスタンスを作成するので、第1引数に矩形波フォースフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Square,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8     g_pDIDevice   =  NULL;
LPDIRECTINPUTEFFECT       g_pEffect      =  NULL;
```

473 サイン（正弦波）パターンの初期化

●DIEFFECT構造体をピリオディックとして初期化してエフェクトを作成してから、サインパターンを指定する　初級

参考サンプルプロジェクト　476　ピリオディックフォース

サインパターンのピリオディックフォースの初期化は、例えば次のようになります。

```
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};

DIPERIODIC periodic={0};
DIEFFECT eff;
```

```
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                    = sizeof(DIEFFECT);
eff.dwFlags                   = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
eff.dwDuration                = INFINITE;
eff.dwSamplePeriod            = 0;
eff.dwGain                    = DI_FFNOMINALMAX;
eff.dwTriggerButton           = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                     = g_dwFFBAxisAmt;
eff.rgdwAxes                  = rgdwAxes;
eff.rglDirection              = rglDirection;
eff.lpEnvelope                = 0;
eff.cbTypeSpecificParams      = sizeof(DIPERIODIC);
eff.lpvTypeSpecificParams     = &periodic;
eff.dwStartDelay              = 10;
g_pDIDevice->CreateEffect(GUID_Sine,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

periodicはピリオディック用の構造体です。DIPERIODIC periodic={0};としてゼロ初期化しているので、初期化直後はフォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（ピリオディックフォースの）サイン波フォースのエフェクトインスタンスを作成するので、第1引数にサイン波フォースフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Sine,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8   g_pDIDevice  = NULL;
LPDIRECTINPUTEFFECT    g_pEffect    = NULL;
```

474 トライアングル（三角波）パターンの初期化

●DIEFFECT構造体をピリオディックとして初期化してエフェクトを作成してから、トライアングルパターンを指定する

初級

参考サンプルプロジェクト　476　ピリオディックフォース

トライアングルパターンのピリオディックフォースの初期化は、例えば次のようになります。

```
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};

DIPERIODIC periodic={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                  = sizeof(DIEFFECT);
eff.dwFlags                 = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
eff.dwDuration              = INFINITE;
eff.dwSamplePeriod          = 0;
eff.dwGain                  = DI_FFNOMINALMAX;
eff.dwTriggerButton         = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                   = g_dwFFBAxisAmt;
eff.rgdwAxes                = rgdwAxes;
eff.rglDirection            = rglDirection;
eff.lpEnvelope              = 0;
eff.cbTypeSpecificParams    = sizeof(DIPERIODIC);
eff.lpvTypeSpecificParams   = &periodic;
eff.dwStartDelay            = 10;
g_pDIDevice->CreateEffect(GUID_Triangle,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

periodicはピリオディック用の構造体です。DIPERIODIC periodic={0};としてゼロ初期化しているので、初期化直後はフォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（ピリオディックフォースの）三

角波フォースのエフェクトインスタンスを作成するので、第1引数に三角波フォースフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Triangle,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8  g_pDIDevice = NULL;
LPDIRECTINPUTEFFECT   g_pEffect   = NULL;
```

475 ソウトゥース（ノコギリ刃）パターンの初期化

●DIEFFECT構造体をピリオディックとして初期化してエフェクトを作成してから、ソウトゥースパターンを指定する

初級

参考サンプルプロジェクト　476 ピリオディックフォース

ソウトゥースパターン（ノコギリ刃パターン）のピリオディックフォースの初期化は、例えば次のようになります。

```
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};

DIPERIODIC periodic={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                  = sizeof(DIEFFECT);
eff.dwFlags                 = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
eff.dwDuration              = INFINITE;
eff.dwSamplePeriod          = 0;
eff.dwGain                  = DI_FFNOMINALMAX;
eff.dwTriggerButton         = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                   = g_dwFFBAxisAmt;
eff.rgdwAxes                = rgdwAxes;
eff.rglDirection            = rglDirection;
eff.lpEnvelope              = 0;
eff.cbTypeSpecificParams    = sizeof(DIPERIODIC);
eff.lpvTypeSpecificParams    = &periodic;
eff.dwStartDelay             = 10;
g_pDIDevice->CreateEffect(GUID_SawtoothUp,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

periodicはピリオディック用の構造体です。DIPERIODIC periodic={0};としてゼロ初期化しているので、初期化直後はフォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（ピリオディックフォースの）ノコギリ刃フォースのエフェクトインスタンスを作成するので、第1引数にノコギリ刃フォースフォースのGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_SawtoothUp,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8   g_pDIDevice = NULL;
LPDIRECTINPUTEFFECT    g_pEffect   = NULL;
```

476 ピリオディックフォースをかける

DIPERIODIC構造体に意図した値を入れてフォースを発生させる　初級

サンプルプロジェクト　476 ピリオディックフォース

本書のFFBサンプルは初期化の時にSetParameterメソッドにDIEP_STARTフラグを指定しているため、初期化後すぐにエフェクト（FFB）がかかるようになっています。ただ、初期化時はそれぞれのエフェクトタイプの構造体をゼロ初期化しているのでフォースは発生しません。

フォースが発生するのはユーザーがスライダーを動かしてからこのUpdateEffect関数が呼ばれたときです。

```
HRESULT UpdateEffect()
{
    //デバイスを有効にする。（有効にしないとエフェクト（FFB）が効かない）
    g_pDIDevice->Acquire();
    //エフェクトのパラメーターを設定する。更新したいパラメーターのみを指定すれば良い
    LONG rglDirection[2]={g_iXForce,g_iYForce};
    DIPERIODIC periodic={0};

    periodic.dwPeriod=100000;//マイクロ秒単位。
    periodic.lOffset=0;
    periodic.dwPhase=0;
    periodic.dwMagnitude= sqrt( (double)g_iXForce * (double)g_iXForce +
           (double)g_iYForce * (double)g_iYForce );

    DIEFFECT eff;
    ZeroMemory( &eff, sizeof(eff) );
    eff.dwSize                 = sizeof(DIEFFECT);
    eff.dwFlags                = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
    eff.cAxes                  = g_dwFFBAxisAmt;
    eff.rglDirection           = rglDirection;
    eff.cbTypeSpecificParams   = sizeof(DIPERIODIC);
    eff.lpvTypeSpecificParams = &periodic;

     if(FAILED(g_pEffect->SetParameters( &eff,DIEP_DIRECTION |
DIEP_TYPESPECIFICPARAMS |
                                            DIEP_START )))
    {
          return E_FAIL;
    }
    return S_OK;
}
```

コードは初期化と同じであることに気付くでしょうか。変更すべきパラメーターは初期化時より少ないのでコードも初期化時より少なくて済みます。

なお、g_iXForce、g_iYForceはINT型のグローバル変数で、ユーザーがダイアログのスライダーを動かしたときに増減する変数です。

コンディション（条件）フォース

477 コンディションフォースとは？

●反力を発生させるフォース

初級

　コンディションフォースは他のフォース3種類とは異なり、最もFFBらしいフォースで、ユーザーがコントローラーにかけている力に反応して反力を発生させるというインタラクティブなフォースです。つまり、他のフォースのように積極的に（悪く言えば一方的に）フォースを発生させるのではなく、ユーザーが加えた力を基に、どの程度の力を返すかを決定します。ユーザーの力が加わった時だけにフォースを発生させるという、本当の反力です。いつ、どのような力を発生させるかは状態・条件（コンディション）次第ということです。積極的に決まった形のフォースを発生させるものではないため、グラフで力を表現することはできません。

478 コンディションフォースの種類について

●条件フォースには4つのパターンある

初級

　コンディションフォースには次の4つのパターンがあります。

①スプリング（ばね）

　オフセットで指定した位置まで、スティックをはじき返すフォースを生みます。文字通り、スティックに見えないスプリングが付いたかのような挙動をします。

　したがってオフセットが原点（ニュートラル位置）だった場合は、オートセンター機能と同義になります。オフセット位置は自由に決められるので、もちろんオートセンターと同じではありません。

②フリクション（摩擦）

　一定の反力を生みます。次のダンパーとも似ていますが、ダンパーのように反力が変化するような複雑なものではなく、一定の摩擦力を生みます。

③ダンパー（緩衝器）

　スティックを動かすと、動く方向にまるで低反発クッションが置いてあるかのように、ゆるやかに反力が加わります。文章で表現することは難しいですが、「ずぼずぼっ」って感じでしょうか。

④イナーシャ（慣性）

　物を動かすとき、その重量が重たいほど、いつまでもその動きを続けようという力が強くかかりますよね。イナーシャは、まるでスティック先端に“おもり”をつけたような効

果を生みます。動かした方向への力を止めても、まだ動こうとするような力です。

　条件フォースにおいて4つのタイプとも、積極的な力ではなくなんらかの作用、つまりユーザーがかける力に対する反作用であることが共通していることに気付くでしょうか？（イナーシャも、何らかの作用があって初めて生まれるものですから）

479 スプリング（ばね）の初期化

DIEFFECT構造体をコンディションとして初期化してエフェクトを作成してから、スプリングタイプを指定する

初級

参考サンプルプロジェクト　483　コンディションフォース

　スプリング反力のエフェクトの初期化は、例えば次のようになります。

```
//スティックの場合、軸は2つ以上なのでパラメーターは要素数2の配列にする
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};
//エフェクト（反力のオブジェクト）を作成
DICONDITION spring={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                  = sizeof(DIEFFECT);
eff.dwFlags                 = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
eff.dwDuration              = INFINITE;
eff.dwSamplePeriod          = 0;
eff.dwGain                  = DI_FFNOMINALMAX;
eff.dwTriggerButton         = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                   = g_dwFFBAxisAmt;
eff.rgdwAxes                = rgdwAxes;
eff.rglDirection            = rglDirection;
eff.lpEnvelope              = 0;
eff.cbTypeSpecificParams    = sizeof(DICONDITION);
eff.lpvTypeSpecificParams   = &spring;
eff.dwStartDelay            = 10;

g_pDIDevice->CreateEffect( GUID_Spring,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

springはスプリング用の構造体です。DICONDITION　spring={0};としてゼロ初期化しているので、初期化直後は反力フォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（コンディションフォースの）スプリング反力のエフェクトインスタンスを作成するので、第1引数にスプリング反力のGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Spring,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8  g_pDIDevice = NULL;
LPDIRECTINPUTEFFECT   g_pEffect   = NULL;
```

480 フリクション（摩擦）の初期化

DIEFFECT構造体をコンディションとして初期化してエフェクトを作成してから、フリクションタイプを指定する

初級

参考サンプルプロジェクト　483 コンディションフォース

フリクション反力のエフェクトの初期化は、例えば次のようになります。

```
//スティックの場合、軸は2つ以上なのでパラメーターは要素数2の配列にする
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};
//エフェクト（反力のオブジェクト）を作成
DICONDITION spring={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                    = sizeof(DIEFFECT);
eff.dwFlags                   = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
eff.dwDuration                 = INFINITE;
eff.dwSamplePeriod             = 0;
eff.dwGain                     = DI_FFNOMINALMAX;
eff.dwTriggerButton            = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                      = g_dwFFBAxisAmt;
eff.rgdwAxes                   = rgdwAxes;
eff.rglDirection               = rglDirection;
eff.lpEnvelope                 = 0;
eff.cbTypeSpecificParams       = sizeof(DICONDITION);
eff.lpvTypeSpecificParams      = &spring;
eff.dwStartDelay               = 10;

g_pDIDevice->CreateEffect(GUID_Friction,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

springはフリクション用の構造体です。DICONDITION　spring={0};としてゼロ初期化しているので、初期化直後は反力フォースが発生しません。

構造体を初期化したら、IDirectInputDevice8:: CreateEffectメソッドによりエフェクトを作成します。

ここでは（コンディションフォースの）フリクション反力のエフェクトインスタンスを作成するので、第1引数にフリクション反力のGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Friction,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8  g_pDIDevice = NULL;
LPDIRECTINPUTEFFECT   g_pEffect   = NULL;
```

481 ダンパー（緩衝器）の初期化

DIEFFECT構造体をコンディションとして初期化してエフェクトを作成してから、ダンパータイプを指定する 初級

参考サンプルプロジェクト 483 コンディションフォース

ダンパー反力のエフェクトの初期化は、例えば次のようになります。

```
//スティックの場合、軸は2つ以上なのでパラメーターは要素数2の配列にする
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};
//エフェクトを作成 コンディションフォースのダンパーパターンとして作成する
DICONDITION damper={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                   = sizeof(DIEFFECT);
eff.dwFlags                  = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
eff.dwDuration               = INFINITE;
eff.dwSamplePeriod           = 0;
eff.dwGain                   = DI_FFNOMINALMAX;
eff.dwTriggerButton          = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                    = g_dwFFBAxisAmt;
eff.rgdwAxes                 = rgdwAxes;
eff.rglDirection             = rglDirection;
eff.lpEnvelope               = 0;
eff.cbTypeSpecificParams     = sizeof(DICONDITION);
eff.lpvTypeSpecificParams    = &damper;
eff.dwStartDelay             = 10;

g_pDIDevice->CreateEffect( GUID_Damper,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

springはダンパー用の構造体です。DICONDITION spring={0};としてゼロ初期化しているので、初期化直後は反力フォースが発生しません。

構造体を初期化したら、IDirectInputDevice8::CreateEffectメソッドによりエフェクトを作成します。ここでは（コンディションフォースの）ダンパー反力のエフェクトイ

ンスタンスを作成するので、第1引数にダンパー反力のGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Friction,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8     g_pDIDevice   =  NULL;
LPDIRECTINPUTEFFECT       g_pEffect       =  NULL;
```

482 イナーシャ（慣性）の初期化

●DIEFFECT構造体をコンディションとして初期化してエフェクトを作成してから、イナーシャタイプを指定する

初級

イナーシャ反力のエフェクトの初期化は、例えば次のようになります。

```
//スティックの場合、軸は2つ以上なのでパラメーターは要素数2の配列にする
DWORD rgdwAxes[2]={DIJOFS_X,DIJOFS_Y};
LONG rglDirection[2]={0,0};
//エフェクトを作成 コンディションフォースのイナーシャパターンとして作成する
DICONDITION damper={0};
DIEFFECT eff;
ZeroMemory( &eff, sizeof(eff) );
eff.dwSize                      = sizeof(DIEFFECT);
eff.dwFlags                     = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
eff.dwDuration                  = INFINITE;
eff.dwSamplePeriod              = 0;
eff.dwGain                      = DI_FFNOMINALMAX;
eff.dwTriggerButton             = DIEB_NOTRIGGER;
eff.dwTriggerRepeatInterval = 0;
eff.cAxes                       = g_dwFFBAxisAmt;
eff.rgdwAxes                    = rgdwAxes;
eff.rglDirection                = rglDirection;
eff.lpEnvelope                  = 0;
eff.cbTypeSpecificParams        = sizeof(DICONDITION);
eff.lpvTypeSpecificParams       = &damper;
eff.dwStartDelay                = 10;

g_pDIDevice->CreateEffect(GUID_Inertia,&eff, &g_pEffect, NULL );
```

g_dwFFBAxisAmtは軸の数で、サンプルではスティックを想定しているので軸の数は2です（ハンドルの場合は1）。

そして、rglDirectionは方向を意味し、これもスティックを想定しているので2つ必要です。要素数が2なのはそのためです。

springはイナーシャ用の構造体です。DICONDITION spring={0};としてゼロ初期化しているので、初期化直後は反力フォースが発生しません。

構造体を初期化したら、IDirectInputDevice8::CreateEffectメソッドによりエフェクトを作成します。

ここでは（コンディションフォースの）イナーシャ反力のエフェクトインスタンスを作成するので、第1引数にイナーシャ反力のGUIDを渡します。

```
g_pDIDevice->CreateEffect(GUID_Inertia,&eff, &g_pEffect, NULL );
```

なお、g_pDIDeviceはダイレクトインプットデバイスインターフェイスポインターで適正に初期化されているものとし、g_pEffectは初期化前のエフェクトインターフェイスポインターであるとします。

```
LPDIRECTINPUTDEVICE8   g_pDIDevice   = NULL;
LPDIRECTINPUTEFFECT    g_pEffect     = NULL;
```

483 コンディションフォースを有効にする

●UpdateEffect関数が呼ばれたときフォースが発生する

初級

サンプルプロジェクト　483 コンディションフォース

本書のFFBサンプルは初期化のときにSetParameterメソッドにDIEP_STARTフラグを指定しているため、初期化後すぐにエフェクト（FFB）がかかるようになっています。ただ、初期化時はそれぞれのエフェクトタイプの構造体をゼロ初期化しているのでフォースは発生しません。

フォースが発生するのは、ユーザーがスライダーを動かしてからこのUpdateEffect関数が呼ばれたときです。

なお、このサンプルのコンディションはダンパータイプとして作成しています。

```
HRESULT UpdateEffect()
{
        //デバイスを有効にする。(有効にしないとエフェクト（FFB）が効かない)
        g_pDIDevice->Acquire();
        //エフェクトのパラメーターを設定する。更新したいパラメーターのみを指定すれば良い
        LONG rglDirection[2]={g_iXForce,g_iYForce};
        DICONDITION damper={0};
        damper.lPositiveCoefficient= sqrt( (double)g_iXForce * (double)g_iXForce +
                (double)g_iYForce * (double)g_iYForce );
        damper.lNegativeCoefficient= damper.dwPositiveSaturation;

        DIEFFECT eff;
        ZeroMemory( &eff, sizeof(eff) );
        eff.dwSize                  = sizeof(DIEFFECT);
        eff.dwFlags                 = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS;
        eff.cAxes                   = g_dwFFBAxisAmt;
        eff.rglDirection            = rglDirection;
        eff.cbTypeSpecificParams    = sizeof(DICONDITION);
        eff.lpvTypeSpecificParams = &damper;

        if(FAILED(g_pEffect->SetParameters( &eff, DIEP_DIRECTION |
    DIEP_TYPESPECIFICPARAMS |
                                                  DIEP_START )))
        {
                return E_FAIL;
        }
        return S_OK;
    }
```

コードは初期化と同じであることに気付くでしょうか。変更すべきパラメーターは初期化時より少ないのでコードも初期化時より少なくて済みます。

なお、g_iXForce、g_iYForceはINT型のグローバル変数で、ユーザーがダイアログのスライダーを動かしたときに増減する変数です。

484 FFBコントローラーを検出する

●コールバック関数を実装したうえでIDirectInput8::EnumDevicesメソッドを実行する 初級

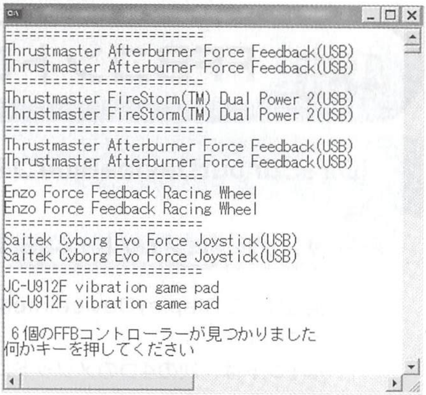
サンプルプロジェクト 484 FFB機器検出

PCに接続されているコントローラーを列挙するにはIDirectInput8::EnumDevicesメソッドを使用します。

第2引数にはアプリケーション定義のコールバック関数を渡します。

第3引数には、FFB機器を列挙したい場合はフラグDIEDFL_FORCEFEEDBACKを含めます。DIEDFL_FORCEFEEDBACKを指定するとコントローラーでもFFB機能が無いものは列挙されなくなります。

▼実行結果



```
==========================
Thrustmaster Afterburner Force Feedback(USB)
Thrustmaster Afterburner Force Feedback(USB)
==========================
Thrustmaster FireStorm(TM) Dual Power 2(USB)
Thrustmaster FireStorm(TM) Dual Power 2(USB)
==========================
Thrustmaster Afterburner Force Feedback(USB)
Thrustmaster Afterburner Force Feedback(USB)
==========================
Enzo Force Feedback Racing Wheel
Enzo Force Feedback Racing Wheel
==========================
Saitek Cyborg Evo Force Joystick(USB)
Saitek Cyborg Evo Force Joystick(USB)
==========================
JC-U912F vibration game pad
JC-U912F vibration game pad

 6 個のFFBコントローラーが見つかりました
何かキーを押してください
```

```
LPDIRECTINPUT8 g_pDI = NULL;
DWORD g_dwCount=0;
/
```

```
//ゲームコントローラー機器の列挙
if(FAILED(g_pDI->EnumDevices(DI8DEVCLASS_GAMECTRL,DIEnumDevicesProc,NULL,
        DIEDFL_ATTACHEDONLY | DIEDFL_FORCEFEEDBACK)))
{
    printf("列挙に失敗しました…");
    return E_FAIL;
}
```

コールバック関数はDirectInput側が実行するものですが、実装はアプリケーション側で行います。

```
//列挙のコールバック関数
BOOL CALLBACK DIEnumDevicesProc( LPCDIDEVICEINSTANCE lpddi,LPVOID pvRef)
{
    g_dwCount++;
    printf("==========================¥n");
    printf("%s¥n%s¥n",lpddi->tszInstanceName,lpddi->tszProductName);

    return DIENUM_CONTINUE;
}
```

コールバック関数の引数LPCDIDEVICEINSTANCE　XXXに情報が入っているので、コールバックされる度にその中身を見れば、デバイス名等を取り出すことが出来ます。

485 FFBコントローラーの能力を取得する

●IDirectInputDevice8の4つのメソッドを使ってあらゆる能力情報を得る 初級

サンプルプロジェクト　485　FFB能力

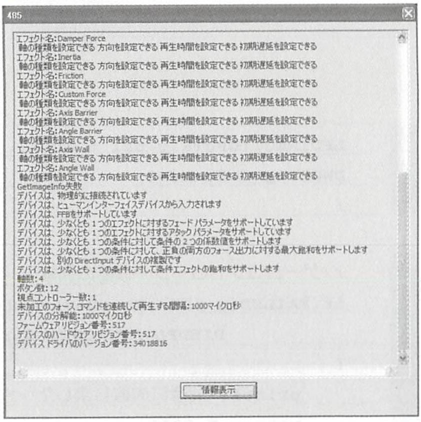
調査は全てDisplayDeviceInfo関数内で行っています。

ポイントは、次の4つのメソッドにより調査しているということです。

調査は、それぞれのメソッドをコールするだけなので特別なことはしていません（コールバック関数が必要なメソッドはコールバック関数を実装してますが）。

なお、MPrintは単なるマクロです。ダイアログに文字を表示するためのコードをいちいち書くのが面倒だったので定義しただけです。

▼実行結果



```
エフェクト名:Damper Force
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Inertia
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Friction
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Custom Force
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Axis Barrier
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Angle Barrier
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Axis Wall
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
エフェクト名:Angle Wall
 軸の種類を設定できる 方向を設定できる 再生時間を設定できる 初期遅延を設定できる
GetImageInfo失敗
デバイスは、物理的に接続されています
デバイスは、ヒューマンインターフェイスデバイスから入力されます
デバイスは、FFBをサポートしています
デバイスは、少なくとも 1 つのエフェクトに対するフェード パラメータをサポートしています
デバイスは、少なくとも 1 つのエフェクトに対するアタック パラメータをサポートしています
デバイスは、少なくとも 1 つの条件に対して条件の 2 つの係数値をサポートします
デバイスは、少なくとも 1 つの条件に対して、正負の両方のフォース出力に対する最大飽和をサポートします
デバイスは、別の DirectInput デバイスの複製です
デバイスは、少なくとも 1 つの条件に対して条件エフェクトの飽和をサポートします
軸数:4
ボタン数:12
視点コントローラー数:1
未加工のフォース コマンドを連続して再生する間隔:1000マイクロ秒
デバイスの分解能:1000マイクロ秒
ファームウェアリビジョン番号:517
デバイスのハードウェアリビジョン番号:517
デバイス ドライバのバージョン番号:34018816
```

IDirectInputDevice8::GetForceFeedbackStateメソッド
IDirectInputDevice8::EnumEffectsメソッド
IDirectInputDevice8::GetImageInfoメソッド
IDirectInputDevice8::GetCapabilitiesメソッド

```
#define MPrint(x) SendDlgItemMessage(g_hDlg,IDC_EDIT1,EM_REPLACESEL,0
,(LPARAM)((TCHAR＊)x))
```

第4部

DirectSetupの極意

DirectX
Visual C++ 2002/2003/2005対応
▶逆引き大全◀
500の極意

DirectX

500 Tips to Use DirectX Better!

第 12 章

486～500

DirectSetupの極意

12-1 DirectSetup（486～500）

486 DirectSetupとは？

●DirectXランタイムにかかわるコンポーネント

初級

DirectSetupは、任意のアプリケーション（あなたが作成したゲームあるいはゲームのインストーラー）内部からDirectXランタイムをインストールすることを目的としたコンポーネントです。また、当該PCにインストールされているDirectXランタイムのバージョンを調べるときにも使用されるコンポーネントです。

「DirectXランタイムをインストール」とはどういうことでしょう？

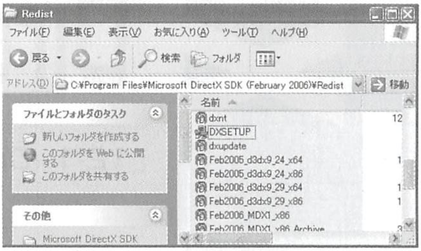
例えば、あなたのゲームがDirectX9を使用して開発されたものとしましょう。そしてそのゲームをプレイすべくインストールしようとしているエンドユーザーがWindows2000を使用している一般の人の場合とします。その人はゲームにあまり興味が無くDirectXをインストールしたことが無い場合、Windows2000にはデフォルトでDirectX8までしか入っていないので、DirectX9を使用するゲームをプレイしてもらうにはDirectX9のランタイムをインストールする(させる)必要があります。

またエンドユーザーがWindowsXPだとしても、OSに最初から入っているDirectXのバージョンからアップデートしていないような環境の場合、通常は動きません。（特に2005年2月以降に開発したものは）初期のWindowsXPの場合、OSに最初から入っているDirectXは8.1です。WindowsXP SP2の場合は9.0cですが、2005年から始まった3ヶ月単位のDirectXアップデートには当然対応できてません。

このような状況なので、意図したランタイムをインストールしてもらわなければ、まず相手側（エンドユーザー側）ではゲームは起動しないと思うのが普通です。

さて、エンドユーザー（ゲームをプレイ、インストールする人）はDXSETUP.exeを起動することによりDirectXランタイムをインストールすることが出来ます。

▼DXSETUP.exe


Redist
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
ヘルプ(H)
戻る
検索
フォルダ
アドレス(D)
C:¥Program Files¥Microsoft DirectX SDK (February 2006)¥Redist
移動
ファイルとフォルダのタスク
新しいフォルダを作成する
このフォルダを Web に公開する
このフォルダを共有する
その他
Microsoft DirectX SDK
名前
dxnt
DXSETUP
dxupdate
Feb2005_d3dx9_24_x64
Feb2005_d3dx9_24_x86
Feb2006_d3dx9_29_x64
Feb2006_d3dx9_29_x86
Feb2006_MDX1_x86
Feb2006_MDX1_x86_Archive

DXSETUP.exeは、再頒布パッケージに同梱されている完結した単体のソフトウェアでマイクロソフトが作成した実行ファイルです。DXSETUP.exeによりエンドユーザーにDirectXランタイムをインストールさせるのであれば、DirectSetupを使用する必要も無ければ知る必要もありません。

一方、DirectSetupは、あくまでも“開発者のプログラム（ゲーム）の内部からDirectXランタイムをインストールするために使用するAPI”であり、外部プログラムであるDXSETUP.exeとは別です。

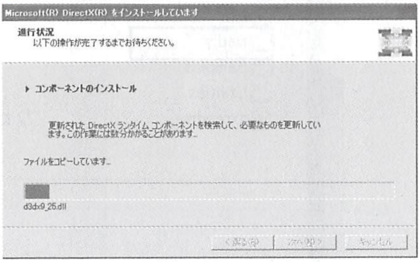
つまり開発者は、エンドユーザーにDirectXランタイムをインストールさせる方法を2種類選択できるということです。

一つは、再頒布パッケージをゲームに同梱し、パッケージ内のDXSETUP.exeをクリックさせて起動させる。したがって、エンドユーザーはゲームとDXSETUP.exeという

2つのプログラムの起動を行うことになります。

二つ目は、エンドユーザーが起動するのはゲームだけ。ゲーム自身がDirectSetupを内部で使用してランタイムのインストールを自動的に行う。ただし、当然この場合でも再頒布パッケージはゲームと一緒に頒布しなくてはなりません。

▼DXSETUP.exeによるインストール画面



Microsoft(R) DirectX(R) をインストールしています

進行状況
以下の操作が完了するまでお待ちください。

▸ コンポーネントのインストール

更新された DirectX ランタイム コンポーネントを検索して、必要なものを更新しています。この作業には数分かかることがあります...

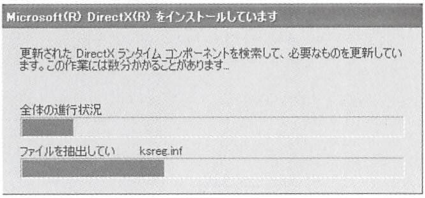
ファイルをコピーしています...

d3dx9_25.dll

最初の方法をとるのであれば、本章を読む必要はありません。

正直筆者としては、この方法で十分だし一番確実だとも思います。この方法をとったとしても、エンドユーザーは最初の一回だけDXSETUP.exeを起動すればいいだけの話です。

▼DirectSetupによるゲーム内からのインストール起動画面



Microsoft(R) DirectX(R) をインストールしています

更新された DirectX ランタイム コンポーネントを検索して、必要なものを更新しています。この作業には数分かかることがあります...

全体の進行状況

ファイルを抽出してい　ksreg.inf

487 再頒布とは？　Redistとは？

再頒布用ランタイムパッケージを使用する

初級

Tips487で述べたとおり、特に2005年2月以降のSDKを使用して開発されたゲームにとっては、まず間違いなくエンドユーザーにランタイムをインストールさせる必要が出てきます。

再頒布とは、DirectXランタイムをマイクロソフトではなく第3者である読者のような開発者が、ゲームに同梱してエンドユーザーに提供（配布）することです。言い換えると二次配布です。

再頒布するための“再頒布用ランタイムパッケージ”が存在しますので、開発者が再頒布するときはそれを使用しなくてはなりません。普通のランタイムを再頒布したりSDKを丸ごと頒布することは禁止されています。また、再頒布パッケージでも、そのなかの一部分だけを配布することは禁止されていますので、気をつけてください。再頒布するときは、必ず再頒布パッケージの全てを再頒布するようにしましょう。

さて、では開発者は再頒布パッケージはどのようにして入手するのでしょう？

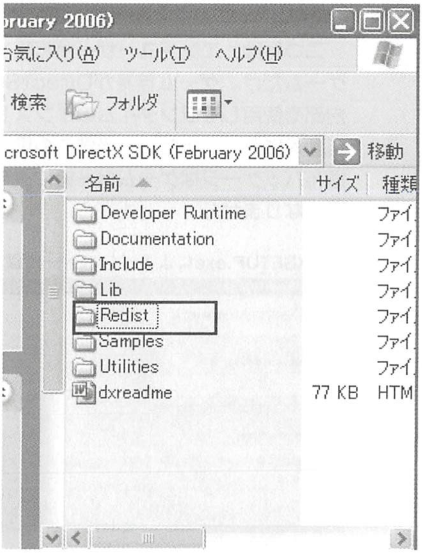
再頒布パッケージはSDKの中にあります。通常（というより今まではずっと）Resdistというフォルダがそうです。

“Redist”は再頒布を意味する英単語redistribution（リディストリビューション）を省略したものなので、同じ意味です。

繰り返し述べますが、Redistフォルダ内の特定一部分のファイルだけを再頒布しては

いけません。必ずRedistフォルダ内の全てのファイルを再頒布してください。

▼実行結果


bruary 2006)
お気に入り(A)　ツール(T)　ヘルプ(H)
検索　フォルダ
crosoft DirectX SDK (February 2006)
移動
名前
サイズ
種類
Developer Runtime
Documentation
Include
Lib
Redist
Samples
Utilities
dxreadme
77 KB
HTM
ファイ

One Point

再頒布パッケージは、マイクロソフトのサイトからも無償でダウンロードできます。ただ、ダウンロードで再頒布パッケージを入手する場合、気をつけなければならないのは、開発に使用したSDKのバージョンと同じRedsitを配布するということです。例えば、April2005で開発したのに、December2005の再頒布パッケージを配布してしまうなど。特に最近は目まぐるしくアップデートが行われているので注意が必要です。確実なのは開発に使用したSDKがインストールされているフォルダ内のRedistフォルダを使用することです。

488 EULAとは？

●「エンド ユーザー使用許諾契約書」のこと

初級

EULAのフルネームは、End-User License Agreementであり、意味は「エンド ユーザー使用許諾契約書」です（ちなみに筆者は「ユーラ」と発音しますが、単に「イー・ユー・エル・エー」と発音する人もいます）。

大抵のソフトウェア製品にはEULAが付いています。ソフトを最初にインストールする際、長々と契約文書がダイアログ等で表示されるのを見たことがあるでしょう。普通は

EULAに同意しなければプログラムをインストールできない（してはいけない）ことになります。

DirectXランタイムをインストールさせる場合には、必ずこのEULAを表示して、エンドユーザーの同意を確認する必要があります。そして、同意しない場合、ランタイムをインストールさせてはいけません。

EULAの表示方法はTips496を参照してください。

489 DirectXランタイムバージョン番号について

●DirectXランタイムのバージョン番号を把握する

初級

DirectXランタイムをインストールする際、まず最初に行うことは、そのエンドユーザーのPCにとって当該ランタイムをインストールする必要があるか無いかということです（大抵は必要あるとは思いますが）。もし運良くゲームに使用しているDirectXと全く同じランタイムがすでにエンドユーザーのPCにインストールされているのであれば、インストールは不要になります。

ランタイムのバージョンはかなり多くあり、次のとおりです。DirectX5以降しか書きませんが、Windows98SE以降のOSであれば、DirectX5以上のランタイムが最初からPCに入っていることになるので、事実上これで十分です。

DirectX5	0x00040005	メジャーバージョン4	マイナーバージョン5
DirectX6	0x00040006	メジャーバージョン4	マイナーバージョン6
DirectX7	0x00040007	メジャーバージョン4	マイナーバージョン7
DirectX8	0x00040008	メジャーバージョン4	マイナーバージョン8
DirectX9	0x00040009	メジャーバージョン4	マイナーバージョン9

バージョン番号は32ビット値になっています。上位16ビットはメジャーバージョンを意味し、下位16ビットはマイナーバージョンを意味します。ほとんどの人はDirectX7とかDirectX9の数字をメジャー番号と思っていたのではないでしょうか？DirectX5以降DirectX9まで、メジャーバージョンは全て4です。

DirectX8.1以降のものは、さらに細かく“リビジョン”番号を持っています。

DirectX 8	無し	
DirectX 8.1	0x0001032A	初期のWindowsXPに同梱されているバージョンです。
DirectX 8.1	0x00010371	
DirectX 8.1a	0x00010385	
DirectX 8.1b	0x00010385	
DirectX 8.2	0x00020386	
DirectX 9.0x	00000384	

DirectX 9a	0x00000385	
DirectX 9b	0x00000386	
DirectX 9c	0x00000387	WindowsXP SP2に同梱されているバージョンです。
DirectX 9c	0x00000388	

リビジョン番号も32ビット値です。したがって、8.1以降DirectXランタイムのバージョンはリビジョンを含めると64ビット値ということになります。

リビジョン番号を含めた完全なバージョン番号は例えばDirectX 8.1 であれば、

バージョン番号0x00040008
リビジョン番号　0x0001032A

なので、それぞれの10進数値を繋げて、4.08.01.0810などと表現されます（32a（8進数）=810（10進数））。

一般式も一応書いておきます。

バージョン上位16ビットの10進数値.バージョン下位16ビットの10進数値.リビジョン上位16ビットの10進数値.リビジョン下位16ビットの10進数値
メジャーバージョン番号.マイナーバージョン番号.メジャーリビジョン番号. .マイナーリビジョン番号

Direct9cの場合、注意が必要です。DirectX9cは2005年2月以降、D3DX（D3D本体ではない）が本体とは別のDLLファイルとして分離されたため、同じDirectX9cでも多くのバリエーションがあります。

例えば、DirectX9c April2005のPCでは、DirectX9c June2005のD3DX系DLL（d3dx9_25.dll）が無いので、June2005で開発したゲームは動きません。

One Point

確認はランタイムのバージョンのみならず、D3DXのバージョンを確認することも必要です（本書サンプルは全ての可能性を確認します）。

490 D3DX系のdllバージョン番号について

D3DXのDLLは細かくバージョン分けされている

初級

DirectX9.0c　February2005から、3DXのDLLは細かくバージョン分けされています。それぞれのバージョン番号は次のとおりです。

February2005	24
April2005	25
June2005	26
August2005	27
December2005	28
February2006	29

これらの番号を基に、プログラム内でどの年月のD3DXかを特定します。たった2、3ヶ月ずれているだけでもDLLファイル名が異なるので、当然相手方でゲームは動かなくなります。

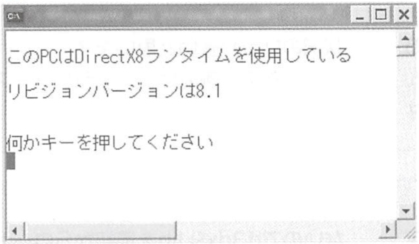
厳密に言うと、プログラム内でD3DX系関数を一切使用していなければ動きますが、そのようなプログラムは稀でしょう。

491 D3DX系のdllバージョン番号を知る方法

自分のPC及び、エンドユーザーのD3DXのバージョンを取得する　初級

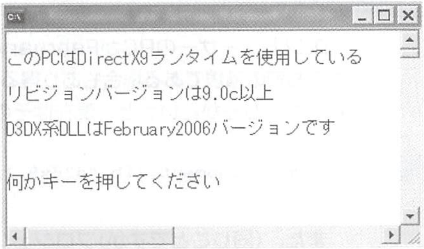
サンプルプロジェクト　491　DXバージョン

▼実行結果



```
このPCはDirectX8ランタイムを使用している
リビジョンバージョンは8.1

何かキーを押してください
```

▼実行結果



```
このPCはDirectX9ランタイムを使用している
リビジョンバージョンは9.0c以上
D3DX系DLLはFebruary2006バージョンです

何かキーを押してください
```

D3DX系コンポーネントは、開発者にとって悩みの種です。2005年2月以降、3ヶ月単位でアップデートがなされているからです。

これはどういうことかと言うと、まず間違いなく、DirectXランタイム（厳密に言えば、その中でD3DX系のDLL）を相手側（エンドユーザー側）でインストールしてもらわないと動かないことを意味します。

開発段階、つまりあなたの環境におけるD3DX系のバージョンを知る方法は簡単です。D3DX_SDK_VERSION（D3DX_VERSIONではない！）という記号定数が定義されていて、その上にマウスポインタを置けばインテリセンス機能で値が見れるはずです。あるいは、d3dxcore.hヘッダーファイルを覗いても知ることができます。

ちなみにFebruary2006のD3DX_SDK_VERSIONは29です。開発段階ではなく、ゲームを配布後、エンドユーザーの環境におけるバージョンをプログラムが知る方法は、コード内でD3DXCheckVersion関数を実行することです。

例えば、D3DXCheckVersion（32,29）とした場合、エンドユーザーのPCにFebruary2006のDLLがインストールされ

ているのであれば関数はTRUEを返します。February2006のDLLが存在しなければFALSEを返します。プログラムはそれによりインストールの必要性を判断できます。

One Point

注意しなければならないことは、先に書いたように、むしろハードコード（生の数字32,29とか）のほうが安全だということです。

礼儀正しく、

D3DXCheckVersion(D3D_SDK_VERSION, D3DX_SDK_VERSION)

のようにマクロで書いてしまうのは危険です。そのコードをコンパイルする時、たまたま別のバージョンのSDKを一時的に使用していたとすると、マクロが意図するものと異なる数字に展開されるからです。その場合、当然February2006を正確に判断できなくなります。

492 そもそもD3DXCheckVersion関数が動かないのでは？

●February2005未満の環境にもチェックプログラムを対応させる方法　初級

エンドユーザーのPCがFebruary2005よりも前の環境である場合もあり得るわけで、しかもどちらかというと、現在はそのような状況のほうが多いと思います。

そのため、例えば次のようなソースコードを書くことはできません。

```
#pragma  comment(lib,"d3dx9.lib")
```

また、（同じことですが）プロジェクト設定でd3dx9.libをロードすることもできません。

このようにしてしまうと、生成されるexeファイルは起動直後にdllを探してしまいます。実行ファイルはDLLが存在しないエンドユーザー上のPCで起動された直後、そのPC上では存在しないd3dx9_xx.dllファイル内の関数をメモリにマッピングしようとするので、exeファイル起動フェーズで引っ掛かります。

この理由から、少なくともチェックプログラムは、あらゆる環境上で動かなくてはならないのでd3dx9.libを最初からロードすべきではありません。

だからと言ってD3DXにリンクしなければD3DXCheckVersion関数が使えません。でもD3DXCheckVersion関数は使いたいという場合、どうしたらいいでしょう？

この問題は、「DLLの遅延ロード」で解決します。実行中、動的にDLLをロードすれば、exe起動フェーズで引っ掛かることもないし、D3DXCheckVersion関数も使えます。

サンプルでは次のようにしています。

```
typedef INT (WINAPI * LPD3DXCHECKVERSION)( DWORD ,DWORD );
static HMODULE hD3DX_DLL=NULL;
hD3DX_DLL=LoadLibrary("d3dx9_29.dll");
```

まず、D3DXCheckVersion関数のポインターを自前に定義します(LPD3DXCHECKVERSION)。

Win32APIのLoadLibrary関数によりd3dx9_29.dllをロードし、DLLのハンドルを得ます。

```
static LPD3DXCHECKVERSION CHECK=(LPD3DXCHECKVERSION)GetProcAddress(
hD3DX_DLL, "D3DXCheckVersion" );
```

DLLのハンドルにより、DLL内の関数(D3DXCheckVersion)のアドレスを取得し、それを用意しておいたLPD3DXCHECKVERSION型のCHECKに代入します。

これ以降は、CHECKがD3DXCheckVersion関数として振舞うことができます。

```
if (CHECK(32, 24))
{
    printf("¥nD3DX系DLLはFebruary2005バージョンです¥n");
}
```

One Point

実は、サンプルではD3DXCheckVersion関数を実質的には使用していません（念のためコード上は実行していますが）。当該バージョンのDLLファイルの有無で、その環境でもD3DXバージョンを判断しています。

493 D3DX_VERSION定数でのチェックは危険！

●D3DX_VERSIONマクロよりも細かくバージョン番号を得る

初級

D3DXのバージョンを調べたり、確認するときは必ずD3DX_SDK_VERSIONマクロ(Tips490、491）を使用しなくてはなりません。

D3DX_VERSIONマクロを手掛かりにしては絶対にいけません。

なぜなら、D3DX_VERSIONはFeb2005からfeb2006まで、全て同じ値(0x0902）だからです。August2005でもDecember2005でもFebruary2006でもD3DX_VERSIONは全て0x0902なのです。

もうお分かりですね。同じ値であるということは、細かいバージョンを区別することが出来なくなることを意味します。

494 再頒布処理に必要なファイル

●意図したDirectXの再頒布に必要なファイルを考える

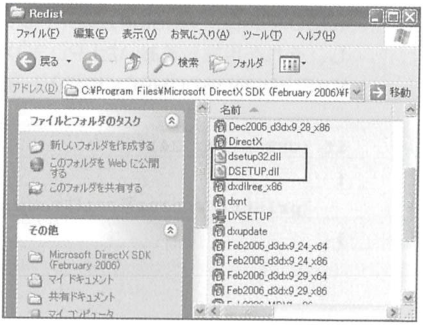
初級

これまで何度も述べたとおり、再頒布するときは再頒布用ファイルの全てを配布しなくてはなりません。逆に言うと、“必要”なファイルをいちいち考える必要がなくなるとも言えます。

再頒布パッケージは、SDKをインストールしたルートディレクトリにおける、Redistフォルダ内にある全てのファイルです。

そして、インストーラーやゲーム内からDirectSetupコンポーネントの関数を使用するためには、dsetup.dllとdsetup32.dllが、プログラムと同じディレクトリか、プログラム内で指定したパス上になければなりません。dsetup.dllとdsetup32.dllもRedistフォルダ内にあります。

▼実行結果


Redist
ファイル(F)
編集(E)
表示(V)
お気に入り(A)
ツール(T)
ヘルプ(H)
戻る
検索
フォルダ
アドレス(D)
C:¥Program Files¥Microsoft DirectX SDK (February 2006)¥F
移動
ファイルとフォルダのタスク
新しいフォルダを作成する
このフォルダを Web に公開する
このフォルダを共有する
その他
Microsoft DirectX SDK (February 2006)
マイ ドキュメント
共有ドキュメント
マイ コンピュータ
名前
Dec2005_d3dx9_28_x86
DirectX
dsetup32.dll
DSETUP.dll
dxdllreg_x86
dxnt
DXSETUP
dxupdate
Feb2005_d3dx9_24_x64
Feb2005_d3dx9_24_x86
Feb2006_d3dx9_29_x64
Feb2006_d3dx9_29_x86

読者は気付いたかもしれませんが、DirectSetupの実態はdsetup.dllとdsetup32.dllです。

495 “DirectXアンインストール不可能”問題について

●セットアップの“動作確認”を円滑に行う方法

初級

DirectXSDKはアンインストールできますが、DirectXランタイムはアンインストールできません。これは間違いありません。

しかし、再頒布セットアッププログラムを作成する開発者であれば、そのプログラムが本当に機能するのか確認する作業が伴いますよね。アンインストールできないと、セットアッププログラムの動作を“たった一回”しか確認できません。場合によっては1回も“確認できません“。理由は想像できますね？一旦上位バージョンをインストールしたら、それ以降は”インストール不要“という判断になるからです。また、開発者の環境が最初から上位バージョンの場合は、確認を一度も行えません（大抵はこの状況でしょう）。

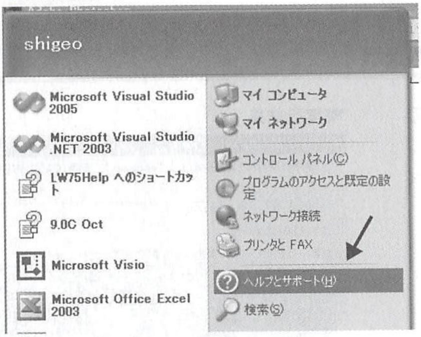
一般的に、何らかのプログラムを作るときは動作確認を数百・数千回行うわけですから、これでは話になりません。

さて何か良い方法はないものでしょうか？

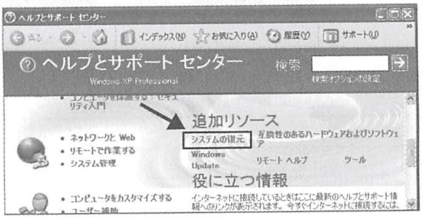
あります。「システムの復元」を上手く利用すればいいのです。ランタイムをインストールする前に、復元ポイントを設定し、その

後でインストールします。セットアップの動作を確認したら、またもとの復元ポイントに戻せば、ランタイムのバージョンは以前のものに戻り、アンインストールと同じ効果を期待できます。

▼実行結果


shigeo
Microsoft Visual Studio 2005
Microsoft Visual Studio .NET 2003
LW75Help へのショートカット
9.0C Oct
Microsoft Visio
Microsoft Office Excel 2003
マイ コンピュータ
マイ ネットワーク
コントロール パネル(C)
プログラムのアクセスと既定の設定
ネットワーク接続
プリンタと FAX
ヘルプとサポート(H)
検索(S)

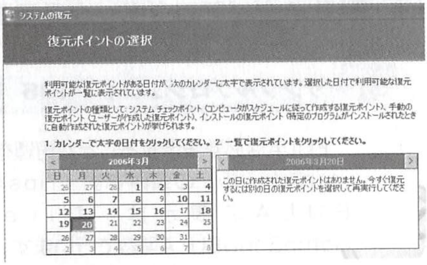
▼実行結果


ヘルプとサポート センター
戻る
インデックス(N)
お気に入り(A)
履歴(Y)
サポート(U)
ヘルプとサポート センター
Windows XP Professional
検索
検索オプションの設定
リティ入門
ネットワークと Web
リモートで作業する
システム管理
コンピュータをカスタマイズする
追加リソース
システムの復元
互換性のあるハードウェアおよびソフトウェア
Windows Update
リモート ヘルプ
ツール
役に立つ情報

インターネットに接続しているときはここに最新のヘルプとサポート情報へのリンクが表示されます。今すぐインターネットに接続するには、

システムの復元は、スタートボタン→ヘルプとサポート→システムの復元で行えます。

▼実行結果


システムの復元
復元ポイントの選択

利用可能な復元ポイントがある日付が、次のカレンダーに太字で表示されています。選択した日付で利用可能な復元ポイントが一覧に表示されています。

復元ポイントの種類として: システム チェックポイント (コンピュータがスケジュールに従って作成する復元ポイント)、手動の復元ポイント (ユーザーが作成した復元ポイント)、インストールの復元ポイント (特定のプログラムがインストールされたときに自動作成された復元ポイント)が挙げられます。

1. カレンダーで太字の日付をクリックしてください。 2. 一覧で復元ポイントをクリックしてください。

2006年3月

日	月	火	水	木	金	土
26	27	28	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

2006年3月20日

この日に作成された復元ポイントはありません。今すぐ復元するには別の日の復元ポイントを選択して再実行してください。

また、システムの復元は5分程度、環境によってはもっと時間がかかるものなので、やはり通常のアプリの動作確認ほどスムースにはいきません。ランタイムのセットアップ関係は、このようなストレスも発生するものです。

One Point

読者の多くはこの解説を読む前に、既に上位バージョンをインストールしていると思われます。復元ポイントを適当な場所に設定していない場合は、少々辛いです。一番良いのは、多少復元ポイントが戻りすぎても気にならないようなセカンドPCがあればいいですね。

496 EULAを表示する

EULAダイアログボックスを表示する

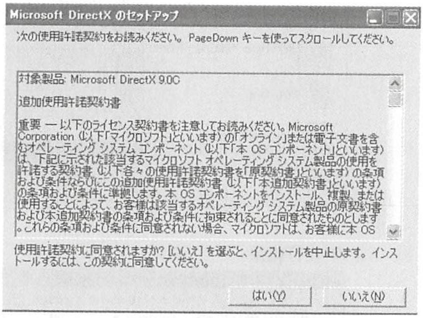
初級

サンプルプロジェクト　496　EULA表示

EULAは必ず表示することが再頒布（後のインストール）の条件です（Tips488）。EULAの表示は、DirectXSetupShowEULA関数で行います。

サンプルでは次のようになります。

▼実行結果



次の使用許諾契約をお読みください。PageDown キーを使ってスクロールしてください。

対象製品: Microsoft DirectX 9.0C

追加使用許諾契約書

重要 ― 以下のライセンス契約書を注意してお読みください。Microsoft Corporation (以下「マイクロソフト」といいます) の「オンライン」または電子文書を含むオペレーティング システム コンポーネント (以下「本 OS コンポーネント」といいます) は、下記に示された該当するマイクロソフト オペレーティング システム製品の使用を許諾する契約書 (以下各々の使用許諾契約書を「原契約書」といいます) の条項および条件ならびにこの追加使用許諾契約書 (以下「本追加契約書」といいます) の条項および条件に準拠します。本 OS コンポーネントをインストール、複製、または使用することによって、お客様は該当するオペレーティング システム製品の原契約書および本追加契約書の条項および条件に拘束されることに同意されたものとします。これらの条項および条件に同意されない場合、マイクロソフトは、お客様に本 OS

使用許諾契約に同意されますか? [いいえ] を選ぶと、インストールを中止します。インストールするには、この契約に同意してください。

```
//EULA表示
if(IDNO==DirectXSetupShowEULA(hWnd))
{
    return  E_FAIL;
}
```

引数としてアプリのウィンドウハンドルをとります。

エンドユーザーがEULAに同意すると（OKボタンを押すと）IDOKが戻り値となり、同意しないと（NOボタンを押すと）IDNOが戻り値となります。

IDNOが返ってきた場合は、それ以降の処理（インストール）を進めてはいけません。

497 結局DirectSetupは必要なの？

使用したほうが好ましい

初級

エンドユーザーの環境におけるバージョンを調べることなく、とにかくDXSETUP.exeをエンドユーザーにクリックさせる手法を採るのであれば、DirectSetupは不用です（バージョンをチェックしたうえで、DXSETUP.exeを起動させる場合にはDirectSetupが必要です）。

DirectSetupの意義、そしてDirectSetupを使用する局面としては次のような場合でしょう。

「エンドユーザーのDirectXランタイムバージョンを調べる必要がある場合」

「カスタムセットアップを実装したい場合」

あるいは、精神的な理由から

「エンドユーザーに、（たとえ最初だけだとしても）2個のプログラムを起動させることに不安を覚える開発者」

または、「“なんか格好悪いなぁ”と思う開発者」

というような感じでしょうか。

筆者が言えることは、DirectSetupは“必ずしも必要ではないが、使用したほうが好ましい”ということです。あとの判断は開発者である貴方次第です。

498 留意事項まとめ

確実にゲームをプレイできる環境を整えてもらうために

初級

ゲームの配布に係る留意事項

エンドユーザーに、確実にゲームをプレイできる環境を整えてもらうことについて、開発者が認識しなければならない事項を再度まとめると次のようになります。

・ほとんどの場合、DirectXランタイムの再頒布が必要になるということ。

・再頒布の方法は3つあること。

①有無を言わさず、とにかく必ずDXSETUP.exeを実行させる。

②DirectXのバージョンを調べる何らかのプログラムによりバージョンを調べてから、必要であればDXSETUP.exeを実行させる。

③DXSETUP.exeを使用せずに、自前のプログラムあるいはゲーム内からランタイムをインストールさせる。

・いずれの方法においても、ランタイムをインストールさせる前には必ずEULAを表示して同意の確認をとること。

・同意しなかった場合は、ランタイムをインストールしない、してはいけないということ。

DirectSetupとの関係

- 3つの方法のうち、①の手法つまりDXSETUP.exeによりランタイムをインストールさせる場合で、かつ、バージョンを調べる必要も無い場合はDirectSetupを使用する必要が無いということ。

- ②の場合は、バージョンを調べる部分でDirectSetupを使用する必要があるということ。そして、本書のサンプルが有用であるということ。

- ③の方法では、バージョン確認はもちろん、インストールにもDirectSetupを使用するので、DirectSetupを熟知しなければならないということ。これにも本書サンプルが有用であるということ。

●再頒布ファイルに係る留意事項

再頒布パッケージに関する留意事項をまとめると次のようになります。

- 再頒布するのは再頒布パッケージのみが唯一許されているということ。普通のランタイムやSDKを再頒布してはいけない。

- 再頒布パッケージは全てのファイルを再頒布すること。一部分のみを再頒布してはいけない。

- 再頒布パッケージはSDK内、Redistフォルダの全てということ。あるいはダウンロードでも入手できる。

- 再頒布パッケージ（dsetup.dll、dsetup32.dllも含む））は開発したSDKのバージョンのものであること。

499　自分のゲームにDirectXセットアップを実装する

●アプリ内でDirectXセットアップを実行する全コード

上級

サンプルプロジェクト　499　セットアップ付きゲーム

最初に言うと、ゲーム内からのセットアップは結構ややこしいです。

自分でコードを書く分には全く簡単なのですが、ややこしいのは解説の順序や内容です。そのため読む側もややこしいかもしれません。正直、「これで本当に伝わるのか？」と思いながら書いているので筆者の精神衛生上よくありません。

このようなコーディングやプロジェクト配置がただ一つの正解ではないので参考程度に見てください。

とは言え、セットアップという特殊な処理からくる制約により、大抵はこのようなコーディングになるとは思います。

まず、ソリューション内に2つのプロジェクトがある理由について解説します。

プロジェクト名「499　セットアップ付きゲーム」の方ですが、これは“ゲーム本体”に見立てています。実際のゲームはもちろんこんなに単純ではありません。ただ、Direct3Dを使用している本体という機能は有しています。

そして、プロジェクト名「セットアップ部分」の方ですが、こちらは文字通りセットアップを行うモジュールです。

2つのプロジェクトからは2つの実行ファ

イルが生成されます。といっても、ユーザーに2つのプログラムを起動させるのであれば、わざわざDirectSetupを使わなくても良いいわけで、再頒布パッケージ内のDXSETUP.exeを起動させることと変わりが無くなってしまいます。

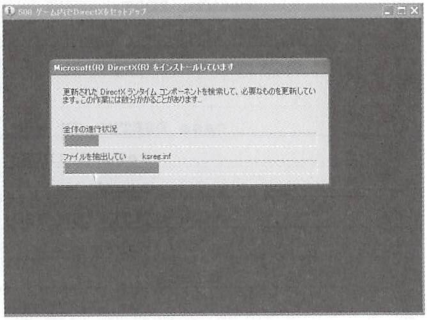
このサンプルでは実行ファイルは2つでも、ユーザーが起動するの実行ファイルは一つだけです。「セットアップ部分」プロジェクトが生成する“セットアップ部分.exe”だけです。“セットアップ部分.exe”は、DirectXを使用できる環境であるか確認あるいはセットアップを行い、DirectXを使用できる環境を整えた後、“セットアップ付きゲーム.exe”を起動します。つまり、“セットアップ付きゲーム.exe”はプログラムから起動されるので、ユーザーが起動するのはあくまで一個セットアップ側だけということです。

このように分ける理由は、DirectXライブラリのロード問題に起因します。

Tips492で述べたとおり、セットアッププログラム自体は全ての環境で動かなければなりません。そのため、ライブラリの初期ロードができません。その結果、LoadLibrary関数で遅延ロードを行い、DLL内から関数アドレスを取得するという面倒なことを行う必要があります。

一方、ゲーム本体のほうでは初期ロードを行います。

▼実行結果



Microsoft(R) DirectX(R) をインストールしています

更新された DirectX ランタイム コンポーネントを検索して、必要なものを更新しています。この作業には数分かかることがあります...

全体の進行状況

ファイルを抽出してい　ksreg.inf

●Main.cpp

```
#pragma  comment(lib,"d3d9.lib")
#pragma  comment(lib,"d3dx9.lib")
/
```

2つに分けた理由がお分かりでしょうか？

なお、読者はこう考えるかもしれません。「ゲーム本体側でも遅延読み込みを行えば、一つの実行ファイルでもいける！」と。

たしかにそれは可能です。

しかし、現実的ではありません。

なぜかと言うと、もしゲーム側でも遅延ロードをしたとしたら、面倒な関数のアドレス取得を“全てのDirectX API”について行わなければなりません（厳密に言えば使用しているAPIの数だけ）。想像してみてください。コード内で使用するDirect3D、DirectSound、DirectInputの全てのAPIについて、Tips492で解説したようなGetProcAddress等を適用することを…目まいがします。どのAPIを使用しているか常に把握すること自体がすでに悶絶ものであり、なにより不恰好です。

さて、次にそれぞれのプロジェクトの内容ですが、セットアップ側（セットアップ部分プロジェクト、SetupMain.cpp）の中身は、491 DXバージョンサンプルと本質的に同じです。

ゲーム側（499 セットアップ付きゲームプロジェクト、SMain.cpp）は…見て分かるとおり、他愛もないものです。解説したくはありません（笑）。

Tips495を参考にして、動作を確認して

ください。ランタイムがインストールされ、ゲーム本体が起動された時、けっこう感動しますよ。

●補足

セットアップのコールバック関数を自前で実装したい場合は、次のようにします。

まずコールバック関数を実装し、

```
DWORD WINAPI SetupCallback( DWORD dwReason, DWORD dwMsgType, LPSTR
strMessage, LPSTR strName,VOID＊ pInfo )
{
    if( strMessage == NULL && strName == NULL )
    {
        return IDOK;
    }
    switch( dwReason )
    {
        case DSETUP_CB_MSG_PROGRESS:
            return IDOK;
            break;
        case DSETUP_CB_MSG_INTERNAL_ERROR:
            break;
        default:
            return IDOK;
    }
    return IDOK;
}
```

DirectXSetupSetCallback関数にコールバック関数のポインターを渡します。

```
HRESULT SetupDirectX(HWND hWnd)
{
    HRESULT hr=0;

    //ライブラリチェック
    HANDLE hSetup = LoadLibrary( TEXT("dsetup.dll") );
    if( hSetup == NULL )
    {
        MessageBox(0,"dsetup.dllがありません","",MB_OK);
        return E_FAIL;
    }
    //EULA表示
    if(IDNO==DirectXSetupShowEULA(hWnd))
    {
        return E_FAIL;
```

```
	}
	//コールバック関数を指定
	DirectXSetupSetCallback(SetupCallback);
```

これで、インストール中のなんらかのアクションは、コールバック関数を経由するので、例えば進行状況などを表示することができるようになります。

500 ゲームクリエイターとして

最も必要なものは「感性の蓄積」

初級

筆者はゲームクリエイターですが、この3年間ずっと書籍の執筆（と専門学校講師）活動を行ってきました。ここで今までの執筆の中で最初の“精神論”を書きたいと思います（そして、もしかしたら最後かもしれません）。

ゲームクリエイターとは過酷な職業です。それはプログラマーやグラフィックアーティストあるいはサウンドクリエイターでも、その道を極めようとしてらっしゃる方であれば同じかもしれませんが、とにかく中途半端な気持ちではやっていけません。もっともアーティスト（絵画とか）よりは多少恵まれているような気はしますが…。

筆者は31歳の時に安定した公務員を退職し、ゲームクリエイター（その当時はまだ修行の身）という“けもの道”に入りました。幸い筆者は運に恵まれましたが、筆者の選んだ道およびアプローチは決して人に勧められるものではありません。ですので、もし同じようなアプローチでゲームクリエイターの道を選ぶ人がいるとしら「絶対やめたほうがいい」と言うでしょう。

しかし反面、そのような人を見たいという気持ちがあるのも事実です。もし読者がそのような状況になったときには、いろいろ話しでもしましょう。

念のため繰り返しますが、筆者はそのような道を勧めているわけではありません。悲惨な状況になっても責任は負いません。そして低くない確率でそうなるとも思います。

ゲームクリエイターにとって最も必要なものは「感性の蓄積」です。技術やお金も大事ですが本質ではありません。教科書を見れば答えが載っているような事柄は、はっきり言って何のアドバンテージにもなりません。アドバンテージを持たなければ、出来上がるゲームはたかが知れてしまいます。お金も全くなければそれは問題ですが、とりあえず生活ができて開発環境を整えるくらいの額があれば我慢できます（我慢しなくてはなりません）。もちろん、それが長引けば問題です。ただし、2年や3年でお金を稼げるようになるなんて夢のそのまた夢です。5年、もしかしたら10年間は我慢しなくてはならないかもしれません。

感性の蓄積とは非常に曖昧な表現ですが、逆に明確な言葉で言い切ってしまうと、それこそ感性を小さな器に閉じ込めてしまうことにもなります。筆者の“感性の蓄積”の1例を挙げると、“アイディアノート”（←センスの無いネーミングと言わないで）があります。10年以上前から、感動したとき、人生の疑

問が一つ晴れたとき、面白い夢を見たとき、危機感を覚えたときなど、その都度忘れないようにノート（電子データですが）に書き込んでいます。イマジネーションが沸いたときはすぐに記録します。イマジネーションは揮発性が高くすぐに消えてしまいますから。

映画なども積極的に観ましょう（お金がなくても）。ただしDVDではなく劇場、しかも多くの観客と一緒に。劇場のほうが何倍も感動が大きいのは、スクリーンや音響もそうですがなによりも多くの人と感動を共有するからだと思います。恋愛もしましょう。苦労も経験しましょう。劣等感も味わいましょう。筆者はこれらに加え耐え難い孤立感も味わいましたが…涙。皆が想像するいわゆる「理想的な開発生活」など最初から出来る訳がありません。

大切なのは、いろいろな状況時に内から突然沸いてくるイマジネーションです。「お勉強」では得られません。貴重なのは機材でもなければ高価なソフトウェアでもありません。貴方の頭に長年蓄積されてきた“データ”なのです。

筆者はそろそろ本格的に本業に戻ります。早ければ3年、遅くて5年後くらいに今世紀最高の“経験”（あえてゲームとは呼びません）を引っさげて皆さんの前に現われるでしょう。そのプラットフォームがPCなのかコンソールなのかは言いませんが、とにかくプロジェクトは本格的に始動しようとしています。

では、その日まで。

（2006年3月20日　鎌田　茂雄）

DirectX

索引

500 Tips to Use DirectX Better!

※表記はTipsの番号です。

数字 Tips No.

1人称視点 ……198
3D効果 ……395
3Dサウンド ……393
3人称視点……197,198

A－D Tips No.

AABB ……244,246
　——メッシュ ……245
D3DCheck Version関数 ……492
D3DX_VERSION定数 ……493
DirectInput時間 ……457
DirectInputオブジェクト ……437
DirectInputデバイスオブジェクト……438,439,440
DirectSetup ……486,497,498
DirectSoundオブジェクト ……324
DirectXアンインストール不可能問題 ……495
DirectXセットアップ ……499
DirectXランタイム ……489
DOF……216

E－T Tips No.

EULA ……488,496
FFBコントローラー ……484,485
FOURCC ……330
FSAA ……211
HRTF ……394
I3DL2リバーブ ……387,388,389
OBB ……247,248
　——アニメーションメッシュ ……249
PCM ……318
Redist ……487
RIFF ……317,429
TPV……197

W－Z Tips No.

WAVE_FORMAT_EXTENSIBLE構造体 ……416
WAVEFORMATEX構造体 ……416
WAVデータフォーマット……417,418
WAVファイル ……314,315,316
　——チャンク ……327
　——判別 ……328
　——開く ……326
　——読み込み ……334
　——マルチチャンネル ……413
Xファイル ……273,274,275,276
　——アニメーション情報 ……281
　——アニメーションメッシュ ……102
　——スキンメッシュ ……111
　——ポリゴン ……280
　——メッシュ ……087,088
Zバッファ……296,309

あ行 Tips No.

アクセス権……453

アニメーションメッシュ……100,101,105

アルファ成分……180

アルファ付きテクスチャ……144,181

アルファブレンディング……178

アルファブレンド……019

アロケート・ヒエラルキー……104

アンビエント光……017

移動……294

- ――音源……396
- ――カメラ……186
- ――テクスチャ……158
- ――プリミティブ……066
- ――メッシュ……091
- ――リスナー……401

イナーシャ……482

陰影……174

インデックスバッファ……257,258

ウェーブス・リバーブ……384,385,386

うねり……361

エコー……375,376,377

エフェクト……357,358

- ――重ねる……363,365
- ――セカンダリーバッファ……363,364
- ――エラー……310,311,312,313
- ――エンベロープ……459
- ――オートセンター……462

オフィスクリーンサーフェイス……300

か行 Tips No.

階層展開……104

回転

- ――音源……397
- ――カメラ……187,194,195
- ――スクリーン平面……032
- ――テクスチャークアッド……049
- ――プリミティブ……067
- ――メッシュ……092
- ――リスナー……402
- ――ワールド空間……039

書き込みカーソル……349,352

拡大・縮小

- ――スクリーン平面……033
- ――テクスチャークアッド……050
- ――プリミティブ小……068
- ――メッシュ……093
- ――ワールド空間……040

重ね順

- ――スクリーン平面……036
- ――テクスチャークアッド……053

カメラ……185

- ――アルファブレンディング……179
- ――ポイントスプライト……117,119

※表記はTipsの番号です。

カメラ位置 ……179
カメラ空間 ……117,119
カリング ……016
カリングトリック ……182
環境光 ……017
環状バッファ ……351
ガーグル ……381,382,383
キーの取得 ……449
（サウンド）キャプチャー ……418,427,428,430
キャプチャーデバイス ……422,424
キャプチャーバッファ ……425
キューブテクスチャ ……170,171
キューブマップ ……169
協調レベル ……325,442,443
金属（の質感） ……137
クオータニオン ……195
（画面の）クリア ……021
クレーン ……191
グーローシェーディング ……125
ゲームクリエイター ……500
ゲームコントローラー ……441,452
コーラス ……366,367,368
コウエフィシエント ……461
交差 ……236
——交点座標 ……235
——直線 ……231
——平面方程式 ……234
——ポリゴン ……233
——ワールド ……232
交点 ……228
交点座標 ……230
（アニメーションメッシュ）交点座標 ……240
固定機能シェーダー ……128,130
コンスタントフォース ……464,465,466
コンディションフォース ……477,478,483
コンプレッション ……369,370,371
ゴム（の質感） ……136

さ行 **Tips No.**

サーフェイス ……307
（音源の）最小・最大 ……398
再初期化 ……045,055,071,097
再生カーソル ……349
再生状態 ……337
再頒布 ……487,494
サインパターン ……473
サウンドカード ……431,432,433
サウンドフォーマット構造体 ……323
サチュレイション ……460
サンプリング ……177,320
シーン ……022
シェーダーの情報 ……305
シェーディングモード ……014
霜 ……132
シャドウボリューム法 ……218,219,220

シャドウマップ法……222,223,224
周波数……340
自由落下……250
初期化……434,435,436,446,447,448
　――Direct3D……007
シリンダー……079,080
深度テクスチャ……225
深度バッファ……296
ズームイン・アウト……192
スキンメッシュ……109,110,112
スクエアパターン……472
（ポイントスプライト）スクリーン空間……118,120
スクリーン平面……030
ステンシルバッファ……297
ストリーミング……344,345,346,347
　――再生……355,356
（アニメーション）スピード……106
（スキンメッシュ）スピード……113
スフィア……075,076
スフィアテクスチャ……168
スフィアマップ……167
スプライト……027,028
スプライトオブジェクト……029
スプリング……479
静的バッファ……338
正方形……150
セカンダリーバッファ……331
　――格納……332
　――再生……335
　――停止……336
　――特定チャンネル……415
　――波形データ……333
　――接線情報……256
全二重……423
ソウトゥースパターン……475
速度
　――音源……399
　――リスナー……403
属性テーブル……263,265,267
属性の活用……268
属性バッファ……262,264,266

た行　**Tips No.**

ダーティ領域……286,287
ダブルステンシル……221
ダンパー……481
（カメラの）注視……188
頂点数……253,277
頂点フォグ……206,209
頂点法線……260,261
（アニメーションメッシュの）直線……236
頂点バッファ……252,255
データ……175,176
ティーポット……083,084
ディストーション……372,373,374

※表記はTipsの番号です。

ディフューズ色 ……173

ディレイ ……359

ディレクション ……458

テクスチャ ……139,140,141

——トラブル ……278

——プリミティブ ……065

——メッシュ ……156

——レンダリング ……299

テクスチャークアッド ……046,047

テクスチャ座標 ……157

——メッシュ ……090

テクスチャステージ ……147

デスクトップ ……306

デバイス消失 ……020

デバイスの取得 ……444,445

デバッグ情報 ……308

トーラス ……081,082

トゥーンシェーディング ……127

透過処理 ……019

トライアングルパターン ……474

ドップラー効果 ……404

ドリーイン・アウト ……193

な行 Tips No.

塗りつぶし ……015,293

は行 Tips No.

跳ね返り ……251

（テクスチャの）反転 ……159

半透明

——スクリーン平面 ……034

——テクスチャークアッド ……051

——プリミティブ ……064

——マテリアル ……138

——メッシュ ……089

——ワールド空間 ……041

半二重 ……423

バージョン番号 ……490,491

バイブレーション ……455

バウンディングスフィア ……241,242,243

バウンディングボックス ……244

バックグラウンド ……343

バックバッファ ……295

バックミラー ……202

バンプマッピング ……165

パーツメッシュ ……237

パーティクル ……121

パラメトリック・イコライザー ……390,391,392

（サウンドの）パン ……339

被写界深度 ……216

歪み ……360

標本化 ……320

ビデオカード ……304

ビデオドライバ……304
ビュー変換……079
ビュー変換行列……199
ビルボーディング
　――カメラ……196
　――プリミティブ……072
　――メッシュ……094
ピクセル
　――コピー……292
　――参照……291
　――変更……290
ピクセルデータ……301,302,303
ピクセルフォグ……210
ピリオディックフォース……470,471,476
ファイル逐次方式……354
フィルター……362
フィルモード……122,123
フォースフィードバック……454,463
フォグ……205,208
フォロートラック……189
フォンシェーディング……126
フォント名……184
複数のメッシュ……096
複数のプリミティブ……070
複数のレンダリング
　――プリミティブ……069
　――メッシュ……095
符号化……322
ふち
　――スクリーン平面……035
　――テクスチャークアッド……052
　――ワールド空間……042
フラットシェーディング……124
フランジャー……378,379,380
フリクション……480
フリップ……026
フルスクリーンアンチエイリアシング……211,212
フレーム階層……103
フロントバッファ……295
ブレンド
　――テクスチャ……148
　――プリミティブ……151,152
　――マルチパス……155
　――メッシュ……153,154
プール……282,283
プラスティック（の質感）……135
プリミティブ……056
プログラマブルシェーダー……129,131
　――テクスチャ……160,162,163,164,166,167
プログレッシブメッシュ……098,099
プロジェクション変換……025
平行移動
　――スクリーン平面……031
　――テクスチャークアッド……048
　――ワールド空間……038
平面投影法……217

※表記はTipsの番号です。

平面方程式 ……227
　――アニメーションメッシュ ……239
ポインター ……350
ポイント ……057,058
ポイントスプライト ……116
ぼかす ……214,215
ボックス ……077,078
ボリューム ……341
ポリゴン ……061,062
　―アニメーションメッシュ ……238
ポリゴン数 ……254,279
ポリゴンの3頂点 ……259

ま行 Tips No.

マウスの取得 ……450,451
マグニチュード ……456
マテリアル ……133,134
　――プリミティブ ……063
マルチチャンネル ……405,406
　――5.1チャンネル ……412
　――再生 ……407,410,411
　――セカンダリバッファ ……414
　――ダウンミックス ……408,409
マルチテクスチャ ……161,146
マルチパス ……145
マルチビュー ……200,201,203,204
回り込みトラック ……190
ミップマップ ……142,143
無限遠平面 ……226
メッシュ ……044,054,073,074
メモリー括方式 ……354
メモリサイズ ……142
モーション ……107,108,114,115
モーションブラー ……213
文字ジオメトリ ……085,086
（情報確認用）文字列 ……183

や行 Tips No.

有限遠平面 ……229
有向バウンディングボックス ……247
読み込みカーソル ……353

ら行 Tips No.

ライト ……018
ライン ……059,060
ラップアラウンド ……348
ランプフォース ……467,468,469
リスナー ……400
リニアPCM ……319,0320
量子化 ……321
隣接性バッファ ……269,270,271,272
累乗サイズ ……149

レンダリング

——ウィンドウ ……008

——ダイアログ ……010,011

——フルスクリーンウィンドウ ……009

——モードレスダイアログ ……012,013

レンダリングターゲット ……298

ローカルメモリ ……342

録音 ……419,4200421

ロック ……284,285,288,289

わ行 Tips No.

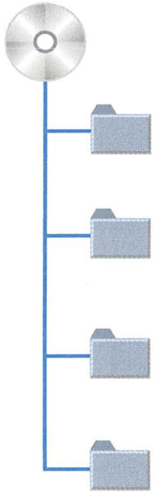
ワールド空間 ……037

ワールド変換 ……023

添付CD-ROMについて

■CD-ROMの構成

添付したCD-ROMには、以下のとおりの構成で各サンプルファイルを収録しています。



Direct3D ── ●第1部で解説したサンプルを収録しています。

DirectInput ── ●第3部で解説したサンプルを収録しています。

DirectSound ── ●第2部で解説したサンプルを収録しています。

DirectSetup ── ●第4部で解説したサンプルを収録しています。

■注意

本書に添付したCD-ROMにはDirectX（SDK）は収録していません。ご入用の際は、以下のマイクロソフトのサイトからダウンロードしてください（無料）。

http://www.microsoft.com/japan/windows/directx/default.mspx

著者紹介

鎌田茂雄（かまだ　しげお）

昭和41年札幌生まれ。高校3年間および“大学”の2年まで、ゲームプログラミングを趣味とする。昭和60年北海学園大学経済学部経営学科入学、2年の春まで、札幌の当時PCゲーム・メーカーでアルバイト。その後、プログラミングからは離れる。平成4年北海学園大学経済学部経営学科卒業、北海道庁に奉職。空知支庁・十勝支庁の経済部建設指導課に勤務。役所で働く中で、ゲームプログラミングを再開。平成11年北海道庁を退職し、フリーのゲームクリエーターとしての準備、活動を行なう。平成14年自作PCゲームをアイディアファクトリー(株)と販売契約を締結し、メジャーリリース。吉田学園情報ビジネス専門学校ゲーム学科で非常勤講師。「ネバーシーン・プロジェクト」(大規模3Dゲーム・プロジェクト)を遂行中。

おもな著書に『はじめての3Dゲーム開発』（工学社)、『ゲームコーディング Vol1～Vol3』（工学社）などがある。

http://www.kamada7.com/

DirectX（ダイレクトエックス）逆引（ぎゃくび）き大全（たいぜん） 500の極意（ごくい）

発行日　2006年 6月 5日　第1版第1刷
　　　　2006年 11月21日　第1版第2刷

著　者　鎌田（かまだ）　茂雄（しげお）

発行者　斉藤　和邦
発行所　株式会社　秀和システム
〒107-0062　東京都港区南青山1-26-1 寿光ビル5F
Tel 03-3470-4947(販売)
Fax 03-3405-7538
印刷所　三松堂印刷株式会社　Printed in Japan

ISBN4-7980-1316-1 C3055

定価はカバーに表示してあります。
乱丁本・落丁本はお取りかえいたします。

本書に関するご質問については、ご質問の内容と住所、氏名、電話番号を明記のうえ、当社編集部宛FAXまたは書面にてお送りください。お電話によるご質問は受け付けておりませんのであらかじめご了承ください。